A Neural Network Based Window Filter and its Training for Image Processing Tasks

A thesis presented in partial fulfilment of the requirements for the degree of Doctor of Philosophy in Technology at Massey University, Palmerston North, New Zealand.

Ralph Harold Pugmire

1995

ABSTRACT

The design and implementation of a neural network based universal window filter (NNWF) is described. Experiments are reported which demonstrate that such a network filter can learn to perform filtering operations from input-target image pairs. Difficulties with training such a filter for more complex tasks are then described. Standard methods of improving the learning performance of neural networks are reviewed. Speculation on development of intelligent systems is presented with particular reference to the purpose and use of logical sequential thought and rules. The importance of an educational environment is outlined and a series of four heuristics for improving the training of neural networks is suggested. Initial experiments with these heuristics are described. The analogy to instructional design theory for humans is proposed and a formal basis for the design of an educational environment for the neural network based window filter is developed. Finally, a series of experiments are reported which test the validity of the use of the educational environment and demonstrate the effectiveness of the methods developed for implementing such an environment for the NNWF.

PREFACE

My father is a psychiatrist, my eldest sister a psychologist, with other brothers and sisters working in medicine and psychiatric nursing it is perhaps natural that I should have an interest in how people think. My undergraduate degree was in Physics and I have worked in various aspects of computing during the last fifteen years. Thus it was with something of a feeling of completing a circle that I discussed the idea of working on neural networks with my supervisor Prof. Bob Hodgson. To combine my interest in the human brain and biological systems with computer science and attempts to make computers more intelligent had great appeal.

The particular avenue for exploring this interaction was that of image processing or computer vision. This I think is a particularly appropriate area for attempting to make use of biological analogies for two reasons; Firstly, animals do it very easily whereas computers find it very difficult. Secondly, computer vision has been dominated by the hard algorithmic approach and by very limited success. What was expected to be easy was found to be very difficult.

"Just determine which is the chair and which is the table, it should be quite straight forward, then ...

If someone could define what is meant by an edge then I might be able to find them, having found them I might be able to find the objects, having found them I might be able to identify them, having ... but so far we cannot adequately define an edge"

So by combining the area of Image Processing, an area in which my supervisors had considerable research experience, with the new and exciting field of Neural Networks we could embark on some interesting work which was based on firm foundations.

ACKNOWLEDGMENTS

I would like to thank my two supervisors, Prof. Bob Hodgson and Dr. Bob Chaplin for their considerable help and encouragement during this process which has been so aptly described as a "research apprenticeship." I have been extremely fortunate in having supervisors, who have encouraged high professional standards in research and writing at the same time as engendering enthusiasm and providing many important insights in the fascinating area we have been studying. They have also allowed me considerable freedom to develop the research in my own way.

I want to acknowledge the considerable amount of "family time" that my wife Sue and my four daughters have sacrificed to enable me to complete this thesis. Without their support I could never have completed this work. Sue has had to act a single parent at times in addition to editing and discussing many drafts and re-drafts of sections of this work. Other members of my extended family have also provided considerable support; my mother with careful proof reading of all of the chapters, my father with insightful discussion on intelligence and other aspects of biological systems, my elder sister Olina with proof reading and encouragement to continue with some of the more speculative aspects of the work. To these and the others who have helped me along the way; thank you.

CONTENTS

Abstract i
Preface ii
Acknowledgements iii
List of Figures viii
Publications x
1. Introduction 1.1
1.1 Scope of Research 1.2
1.2 Thesis Overview 1.3
1.3 Content by Chapter 1.4
1.4 References 1.6
2. Background 2.1
2.1 Image Processing and Window Filters 2.2
2.1.1 Linear Filters 2.4
2.1.2 Sobel Filter 2.4
2.1.3 Marr Hildreth Filter 2.5
2.2 Neural Networks and Back-Propagation 2.9
2.2.1 Back Propagation 2.12
2.2.2 Derivation of Back Propagation Equations 2.14
2.3 Instructional Design 2.19
2.4 Discussion 2.21
2.5 References 2.23
3. A Universal, Neural Network Based, Window Filter 3.1
3.1 Structure & Implementation of the NNWF 3.2
3.1.1 Structure of the NNWF 3.2
3.1.2 Implementation of the NNWF 3.4
3.2 Window Filter Emulation with the NNWF 3.6
3.2.1 Vertical Edge Filter 3.8
3.2.2 Sobel Filter 3.11
3.2.3 Marr-Hildreth Filter 3.14

3.2.4 Effects of Network Parameters 3.17
3.3 Further Experiments with the NNWF 3.21
3.3.1 Using the NNWF to remove noise 3.21
3.4 Improving the performance of the Neural Network Window Filter 3.26
3.4.1 Improvements to the NNWF for Image Processing 3.26
3.4.2 Machine Learning and the NNWF 3.28
3.4.3 Other Neural Network Techniques and the NNWF 3.29
3.5 Discussion 3.37
3.6 References 3.41
4. Neural Networks and Intelligent Systems 4.1
4.1 Introduction 4.2
4.2 Background 4.3
4.3 An Alternative Approach to Intelligence 4.4
4.3.1 Creatures of a fixed design 4.4
4.3.2 Creatures with genetic inheritance. 4.5
4.3.3 Creatures that respond to conditioning. 4.5
4.3.4 Creatures with logical sequential thought 4.6
4.4 The Frame Problem 4.8
4.5 Expert Systems 4.11
4.6 Summary 4.13
4.7 Final Afterthoughts 4.14
4.8 Conclusion 4.16
4.9 References 4.17
5. An Educational Environment 5.1
5.1 Introduction 5.2
5.2 Rationale for the four Heuristics 5.3
5.2.1 Learning simpler similar tasks first 5.3
5.2.2 Learning to perform useful sub-tasks first. 5.3
5.2.3 Using rules provided by a teacher during training. 5.3
5.2.4 Structuring the learning experience. 5.4
5.3 Fuzzy Targets For Training 5.6
5.4 Experiments Using The Four Heuristics 5.7
5.4.1 Task 1: Improving the Marr-Hildreth emulation 5.8
5.4.2 Task 2: Finding E's in images of printed text 5.11
5.4.3 Task 3: Identification of valve parts 5.15
5.5 Discussion 5.18
5.6 References 5.19

6. Instructional Design and Training Artificial Neural Networks 6.1
6.1 What is Instructional Design 6.3
6.2 Differences between ID for people and ID for ANNs 6.6
6.3 Theories of ID and their relevance to ANNs 6.9
6.3.1 Early Theories of ID (Gagné & Briggs) 6.9
6.3.2 A Behavioural Approach (Gropper) 6.13
6.3.3 Component Display Theory (Merrill) 6.24
6.3.4 Elaboration Theory (Reigeluth & Stein) 6.32
6.4 Discussion of ID theories 6.36
6.5 A theory of ID for ANNs 6.37
6.6 References 6.40
7. Training Heuristics and their Implementation 7.1
7.1 Introduction 7.2
7.2 Consideration of Stimulus / Response Characteristics 7.5
7.3 Internal Heuristics 7.9
7.3.1 Learning Useful Sub-Tasks First 7.9
7.3.2 Learning Simpler, Similar Tasks First 7.10
7.3.3 Use of Rules To Assist Learning 7.11
7.4 Implementation Methods for Internal Heuristics 7.15
7.4.1 Eight Methods for Implementing Internal Heuristics 7.16
7.4.2 Methods for implementing sub-tasks 7.24
7.4.3 Methods for Simpler Similar Tasks 7.25
7.4.4 Methods for using Rules 7.26
7.5 Structured Training 7.27
7.6 Summary & Discussion 7.28
7.7 References 7.29

8. Experiments with an Educational Environment for the NNWF 8.1
8.1 Introduction 8.2
8.2 Experiments. 8.4
8.2.1 Sub-task Experiments 8.4
8.2.1.1 Experiment 1.1 Texture as a sub-task 8.7
8.2.1.2 Experiment 1.2 Edges as a sub-task 8.18
8.2.1.3 Discussion of Sub-task experiments 8.20
8.2.2 Finding Tree edges 8.21
8.2.2.1 Consideration of S-R Characteristics 8.29
8.2.2.2 Sub-tasks 8.31
8.2.2.3 Simpler-Similar Tasks 8.35
8.2.2.4 Use of Rules 8.36
8.2.2.5 Structured Training 8.37
8.2.2.6 Discussion of Tree edge experiments 8.44
8.3 Discussion 8.46
8.4 References 8.47
9. Summary & Conclusions 9.1
9.1 Summary 9.2
9.2 Conclusions 9.4

Appendix A: Marr-Hildreth Operator Examples A1
Appendix B: Aspirin/Migraines Additions and Unix Control Files B1
Initialisation File B2
Learn Control File B6
Run Control File B7
Dump Weights Control FIle B8
Appendix C: NIH Image "Plug-ins" and User Routines C1
Marr-Hildreth Plugin C2
Utility Routines C13
User.p From NIH Image C17

List of Figures

Figure 2.1: Operation of a window filte 2.3
Figure 2.2: Commonly used linear filter masks 2.3
Figure 2.3: Marr-Hildreth Edge Filter 2.7
Figure 2.4: Marr-Hildreth Edge Filter Option Effects 2.8
Figure 2.5: Back propagation network 2.13
Figure 3.1: Topology of Neural Network Window Filter 3.3
Figure 3.2: Microscope Test Image 3.7
Figure 3.3: Einstein Test Image 3.8
Figure 3.4: Convergence for Linear Filter Emulation 3.9
Figure 3.5: Emulation of Linear Vertical Edge Filter 3.10
Figure 3.6: Convergence for Sobel Emulation 3.11
Figure 3.7: Emulation of Sobel Edge Filter 3.13
Figure 3.8: Convergence for Marr-Hildreth Emulation 3.16
Figure 3.9: Emulation of Marr-Hildreth Operator 3.17
Figure 3.10: Convergence of NNWF for Large number of training epochs 3.18
Figure 3.11: Effect of Gain and Momentum terms on convergence 3.19
Figure 3.12: Effect of the number of hidden units on convergence 3.20
Figure 3.13: Image degradation with different types of noise 3.22
Figure 3.14: Noise removal with NNWF and median filter 3.25
Figure 5.1: Weight values for sub networks 5.9
Figure 5.2: Combined network structure 5.9
Figure 5.3: Output of combined NNWF 5.10
Figure 5.4: Image of a section of a newsletter 5.12
Figure 5.5: Single word enlarged to show effects of digitisation 5.13
Figure 5.6: Identification of machined parts 5.15
Figure 5.7: Rule generated input and target data 5.16
Figure 6.1: Merrill's Performance - Content Matrix 6.24
Figure 6.2: CDT Content categories 6.26
Figure 6.3: Primary Presentation Forms (PPFs) 6.27

Figure 6.4: Secondary Presentation Forms (SPFs) 6.28
Figure 6.5: Secondary presentations 6.29
Figure 6.6: PPF principles for FIND and USE 6.30
Figure 6.7: Secondary presentation form principles 6.31
Figure 7.1: Solution Graph 7.10
Figure 7.2: Implementation of Internal Heuristics 7.15
Figure 7.3: Use of an embedded function 7.22
Figure 8.1: Images from textured shapes experiments 8.6
Figure 8.2: Texture as a sub-task 8.7
Figure 8.3: H0M0 Standard Training Using Examples 8.8
Figure 8.4: H2M1 Sub-Task Implemented as a Sequence 8.9
Figure 8.5: H2M2 Sub-Task Implemented as Pre-training 8.10
Figure 8.6: H2M3 Sub-Task Implemented as Dual Inputs 8.11
Figure 8.7: H2M4 Sub-Task Implemented using Dual Outputs 8.12
Figure 8.8: H2M5 Sub-Task Implemented as an Embedded Sub-network 8.13
Figure 8.9: Embedded sub-network 8.13
Figure 8.10: H2M6 Sub-Task Implemented as an Embedded Sub-algorithm 8.15
Figure 8.11: Smoothed Target and Best result from NNWF 8.16
Figure 8.12: Target image for edges sub-task 8.19
Figure 8.13: Target image for edges sub-task 8.19
Figure 8.14: Image from Pine Forest 8.22
Figure 8.15: Alternative edge filters 8.24
Figure 8.16: Output from trained NNWF 8.25
Figure 8.17: Training sub-image and target 8.26
Figure 8.18: Intensity profile through training sub-image 8.27
Figure 8.19: NNWF Result from standard training 8.28
Figure 8.20: Separate left right training 8.30
Figure 8.21: Output from "find bark" sub-network 8.31
Figure 8.22: Output from network pre-trained on "find bark" 8.32
Figure 8.23: A gap-filling network 8.33
Figure 8.24: Post processing with gap-filling network 8.34
Figure 8.25: A simpler similar task 8.35
Figure 8.26: Training set extended by use of a rule 8.36
Figure 8.27: Structured training 1 ..... 8.38
Figure 8.28: Combined result 8.39
Figure 8.29: A different forest scene 8.40
Figure 8.30: Edges found by NNWFs 8.41
Figure 8.31: Results from post processing network 8.42
Figure 8.32: Post processing of rule extended result 8.43

PUBLICATIONS

Publications and presentations prepared during the course of the research for this thesis include;

- R.H. Pugmire, R.M. Hodgson, *Why you should know about Neural Networks*, in proceedings of the 27th New Zealand National Electronics Conference NELCON'90, Wellington, 1990
- R.H. Pugmire & R.M. Hodgson, *Alternative Approaches to Cervical Smear Screening*, in proceedings of 5th New Zealand Image Processing Workshop, Palmerston North, August 1990, pp29-33
- R.H. Pugmire, R.M. Hodgson, *Experiments with a Neural Network Based Window Filter*, in proceedings of 5th New Zealand Image Processing Workshop, August 1990, pp48-55
- R.H. Pugmire & R.M. Hodgson, *Watching a Neural Network Learn*, in proceedings of 6th New Zealand Image Processing Workshop, Lower Hutt , 1991, pp27-33
- R.H. Pugmire, *The Purpose of Logical Sequential Thought*, in proceedings of New Zealand Transputer Users Group Meeting , Auckland, 1992, pp53-58
- R.H. Pugmire, *Experiments with a Neural Network Based Window Filter;Hierarchical Neural Networks*, The Fourth Australian Conference on Neural Networks ACNN'93, Melbourne, 1993
- R.H. Pugmire, R.M. Hodgson, R.I. Chaplin, *Rules and Neural Networks*, First Australian and New Zealand Conference on Intelligent Information Systems ANZIIS'93, Perth Australia, 1993
- R.H. Pugmire, R.M. Hodgson, R.I. Chaplin, *Teaching a Neural Network based Window Filter to Perform Difficult Tasks*, in proceedings of the first New Zealand conference on Image and Vision Computing IVCNZ'93, Auckland, 1993, pp
- R.H. Pugmire, R.M. Hodgson, R.I. Chaplin, *Educating a Neural Network for Image Processing*, in proceedings of the first New Zealand conference on Artificial Neural Networks and Expert Systems ANNES'93, Dunedin, November 1993, pp86-91
- R.H. Pugmire, R.M. Hodgson, R.I. Chaplin, *The properties and training of a neural network based universal window filter*, The fifth IEE International Conference on Image Processing and its Applications IPA'95, Edinburgh, July 1995

Chapter 1

1. INTRODUCTION

In this chapter the scope and logical structure of the research work is outlined. The key ideas to be presented in the body of the thesis are introduced and the contents of each chapter is briefly outlined.

1.1 SCOPE OF RESEARCH

There are a wide range of neural network paradigms which could be applied in many different aspects of machine vision and image processing. In this thesis a neural network based window filter is developed and then used as a vehicle for investigating some of the more general aspects of the application of neural networks to image processing. As the research progressed the improvement of the network learning performance by the use of an "educational environment" became a key issue. Thus a major part of the thesis focuses on the rationale behind the design and use of such an educational environment, the analogy to instructional design theory for humans and possible methods for the incorporation of these techniques into the training system of the neural network window filter.

The focus of most current neural network research is on improving the ability of a network to learn. In contrast, in this work we have focused on providing a better learning environment. This is not to deny the importance of the learning methods but only to point out that it is only one of several aspects worthy of study. The term *education* is used in a rather restricted way in this thesis. This is necessary because the more natural term *training* is commonly used, in neural network literature, to refer to simple repetitive presentation of examples.

One aspect of image processing has been investigated, that is the use of a class of local operators, referred to as window filters. In the main a single neural network paradigm has been investigated, that is the multi-layer network with back-propagation learning. A number of methods of improving the performance and utility of this type of network are developed in the thesis.

1.2 THESIS OVERVIEW

The key achievements claimed for this work are;

- A neural network based window filter (NNWF) has been implemented which learns the required transformation from pairs of images. This was first implemented as an extension to the VIPS image processing package [Bailey & Hodgson 1988] and later as an extension to the Aspirin/Migraines [Leighton 1991] package running under Unix.

- It has been demonstrated that this NNWF could learn to emulate standard window filter operations. The importance of this result is not that the operations are performed by a network because, unless the NNWF was implemented in hardware, the standard window filter operations would be performed more quickly by traditional algorithmic techniques. The importance lies in the fact that the operations were learnt, and hence defined, by example rather than by an algorithmic definition.

- Consideration of why the NNWF provided poor performance in some situations and how this performance could be improved led to a consideration of intelligent systems. This in turn led to an important insight into the use of rules by people and an alternative evolutionary purpose for logical sequential thought processes. The idea presented here is that sequential rule based thought may be primarily important as a way of transferring knowledge rather than as a method of determining actions or solving problems.

- The attempt to link these ideas to actual experiments on improving performance of the NNWF led to the use of an analogy to instructional design for people. A study of the main concepts and techniques used in instructional design for people has been made. On this basis five heuristics have been formulated to improve the training of artificial neural networks and a set of methods for applying these heuristics has been implemented for the NNWF.

- The experiments described in the thesis have shown that significant improvements in the training of a neural network based window filter can be achieved by the use of the proposed heuristics.

1.3 Content by Chapter

In chapter two some necessary background is provided by explaining what window filters are and how they are used in image processing, what artificial neural networks(ANNs) are and the operation of the particular type of ANN used in this study. The topic of instructional design for people is also briefly introduced, this is used in the formation of an educational environment discussed in later chapters.

In chapter three the initial implementation of the neural network based window filter is described. Experiments are reported which were used to determine if it could be trained to perform standard filtering operations. This chapter also contains discussion of methods of improving the performance of the NNWF. The types of improvement possible are discussed and the standard methods of improving back propagation learning are outlined. While some previous research was found that used aspects of the techniques developed later in the thesis, no work was found that formally considered the design and use of an educational environment for improving the performance and utility of neural networks.

Chapter four contains the most speculative aspect of this thesis. In this chapter the purpose of logical sequential thought, the development of artificial intelligence and in particular the function and use of rule based thinking is discussed. This speculation includes an alternative suggestion for the function of logical sequential thought and led to the idea of developing an educational environment for training artificial neural networks.

In chapter five the initial outline of four heuristics which could be used to form an educational environment is described. Some initial experiments which test the effectiveness of applying these four heuristics to the NNWF are then reported.

In chapter six the field of instructional design for people and its application to artificial neural networks is discussed. The aim of this chapter is to provide a more formal basis for the design of an educational environment for training artificial neural networks.

In chapter seven an updated set of five heuristics for training artificial neural networks is developed. A set of methods for implementing the heuristics, with particular reference to the NNWF, is developed in this chapter.

In chapter eight a number of experiments are reported which are intended to test the validity and hopefully demonstrate the effectiveness of the educational environment methods proposed in chapter seven.

Discussion and references sections are included at the end of each chapter as this simplifies location of referenced work. The references are listed alphabetically, based on the surname of the first author. The chapter and page number of first citation for each reference is shown to the right of each reference.

1.4 REFERENCES

page first cited

Bailey & Hodgson 1988

D.G. Bailey, R.M. Hodgson, *VIPS - A digital image processing algorithm development environment*, Image and Vision Computing, Vol. 6 No. 3, August 1988, pp176-184 ..... 1.3

Leighton 1991

R.R. Leighton, The Aspirin/MIGRAINES Software Tools: User's Manual, The MITRE Corporation, 1991 ..... 1.3

Chapter 2

2. Background

This thesis is the result of study, and the perhaps unexpected confluence, of material from three branches of knowledge; image processing (specifically window filtering techniques), artificial neural networks (particularly multi-layer networks with back propagation learning), and educational theory (specifically instructional design for people). In this chapter material from each of these areas is introduced to set the scene for the description of a neural network based window filter and an investigation into methods of improving both its performance and the range of situations in which it can be used. The performance and the utility of the neural network window filter have been improved by the design and use of an educational environment.

2.1 Image Processing and Window Filters

The term **image processing** refers to the processes involved in the modification of images to enhance their appearance or facilitate the extraction of information. The term **image analysis** is used to refer to the actual extraction of information from images [Jain 1989, Freeman 1986]. While the examination of animal vision systems would suggest that advantages can be gained by combining processing and analysis [Schwartz 1977, Overington 1992], the work described here primarily relates to image processing.

There are two main reasons for processing images; firstly the image may be enhanced to improve or change its appearance. This may include both cosmetic and informational changes. Examples are; removal of noise or other distracting artifacts, modifications to enable display on a particular medium (television, newspaper, etching) and increasing colour saturation to make a picture more arresting. Secondly, the processing may be carried out to enable subsequent measurement or information extraction. Examples are; removing noise, extracting or enhancing edge information, highlighting features of particular interest etc. Operations used for image enhancement and pre-processing include frequency domain operations such as Fourier analysis, statistical operations, and the use of transformation operators and various filters. Window filters are a type of local operation used in the early stages of many image processing algorithms. They are extensively used for reducing noise, enhancing contrast, edge detection and for producing cosmetic effects such as embossing.

Specifically the term "window filter" refers to the class of local operators in which the output at each position in an output image is defined as a function of the pixels in a "window" surrounding the equivalent position in the input image. The window filter can be thought of as a unit that has multiple inputs and a single output and is scanned across the input image to build up an output image. The window may be any shape and size but square 3x3 windows are the most common. The operation of a window filter is depicted in Figure 2.1. The numerical masks for some simple window filters are shown in Figure 2.2.

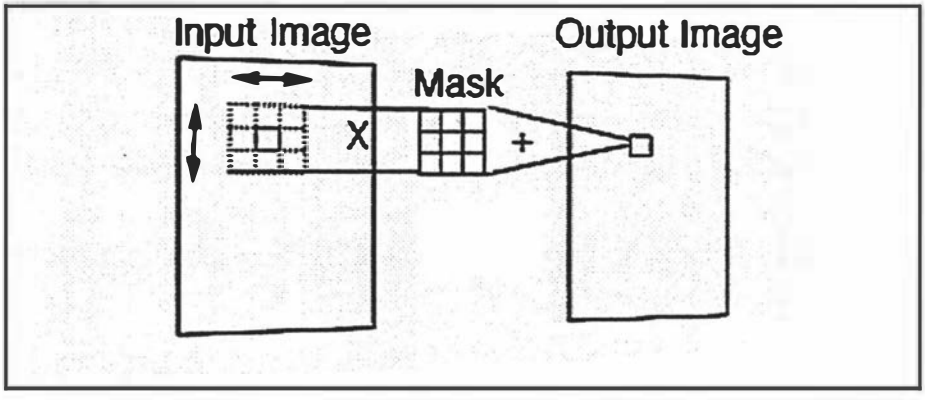


Figure 2.1: Operation of a window filter

Window filters can be based on simple linear functions, non-linear functions or complex rule based functions. One representative filter was selected from each of these three categories for use in the series of experiments described in chapter three. The filters chosen were; the linear vertical edge filter (LVE), the Sobel filter (SF) and the Marr-Hildreth (MH) filter. The LVE filter is a simple linear window filter [Russ 1992], the Sobel filter is a non-linear filter used for simple edge detection [Russ 1992 ibid. p125] and the MH filter is a complex edge detection filter based on a model of the early stages of the human visual system [Marr 1980]. A detailed description of the operation of each of these three filters is given below.

A)
1	1	1
1	1	1
1	1	1

B)
1	0	−1
1	0	−1
1	0	−1

C)
−1	−1	−1
−1	8	−1
−1	−1	−1

D)
−1	0	1
−2	0	2
−1	0	1

Figure 2.2: Commonly used linear filter masks

A) Noise reducing filter
B) Vertical edge filter
C) Omnidirectional edge filter
D) Mask for Sobel filter

2.1.1 Linear Filters

In a linear filter a mask is convolved with the input image, each output pixel being the sum of the products produced by multiplying each mask element by the image pixel in the equivalent position, relative to the centre of the mask. If we refer to the mask elements as $m_{s,t}$, the input image pixels as $i_{x,y}$ and the equivalent output pixels as $o_{x,y}$ then the operation of the filter is given by

$$o_{x,y} = \sum_{t=-1}^{+1} \sum_{s=-1}^{+1} i_{x-s,y-t} m_{s,t} \text{ for a 3x3 window.}$$

For simplicity the filters are normally implemented with the mask index signs reversed as below. Therefore for a true convolution for non symmetrical masks the mask should be reflected about the central pixel in both the x and y directions before the filter is run.

$$o_{x,y} = \sum_{t=-1}^{+1} \sum_{s=-1}^{+1} i_{x+s,y+t} m_{s,t} \text{ (normal implementation)}$$

The result of the convolution will, in general, contain signed values over an extended range. In practice either an offset is used to allow negative numbers to be displayed or only magnitude is preserved. The output from the filter is then scaled to produce a full range of grey scales [Russ 1992 ibid. p125].

2.1.2 Sobel Filter

The Sobel edge detection filter is an example of a non-linear filter formed by a combination of linear filters. The output of the Sobel filter is the RMS combination of the outputs of two linear filters one using the mask

−1	0	1
−2	0	2
−1	0	1

and the other using the transpose of this mask. The equation for each output pixel is thus

$$o_{x,y} = \sqrt{\frac{\left(\sum_{s=-1}^{+1} \sum_{t=-1}^{+1} i_{x+t,y+s} m_{t,s}\right)^2 + \left(\sum_{s=-1}^{+1} \sum_{t=-1}^{+1} i_{x+t,y+s} m_{s,t}\right)^2}{2}}$$

The resulting real valued image can either be scaled back into the original pixel range or truncated. The Sobel filter provides a measure of two-dimensional gradient by combining the output of two linear edge filters, one performing directional differentiation in the horizontal direction and one in the vertical direction.

2.1.3 Marr Hildreth Filter

Marr & Hildreth suggested that the purpose of early visual processing is to construct a primitive but rich description of a visual scene which can be used to determine the reflectance and illumination of visible surfaces, and their orientation and distance relative to the viewer. On this basis they developed an edge-detection filter which extracts zero-crossings after convolving the image with a Laplacian of Gaussian mask. This has become known as the Marr-Hildreth operator or edge filter [Marr & Hildreth 1980 ibid.].

The MH operator used for this work used the following implementation. A mask is calculated using the formula $\nabla^2 G(x,y) = \frac{1}{\pi\sigma^4}\left(1 - \frac{\left(x^2+y^2\right)}{2\sigma^2}\right)e^{-\left(x^2+y^2\right)/2\sigma^2}$ which is then convolved with the input image to produce a real valued image. The value of sigma in this equation effectively determines the width of the ideal mask.

Spurious zero crossings were removed by setting any resultant pixels with a magnitude less than a threshold to zero. This region around zero is referred to as the deadband. Zero crossing points are then found by examining opposite pairs of the eight surrounding pixels at each position. If a sign change is found then an edge is considered to be present. Two output alternatives are provided;

1. If a zero crossing is found an output is produced in which the rate of change at the zero crossing is represented as a grey scale.

2. Alternatively, the inverse of the distance from the actual zero crossing is calculated and this is represented as a grey scale.

At each position the result of the convolution for that pixel(c) and the eight surrounding pixels is considered. The four pairs of pixels on either side of c are examined to look for a change of sign. If a change of sign is found then a value s is calculated which is the largest difference between pixels on either side of c. The output value z is then calculated as

$z = s - c$ for scale by rate of zero crossing

$z = 1 - \frac{c}{s}$ for scale by distance from zero crossing point

This results in pixels which occur exactly on a zero crossing being shown as black. Where the zero crossing occurs between two pixels both pixels have a grey level representing their relative distance from the zero crossing. This provides a grey level image which is a form of anti-aliased edge strength map to

sub-pixel resolution. For both output forms the resultant edge image is scaled to use the full range of the 8 bit grey scale representation. Another common approach is to provide an output image of greater resolution than the input image to enable the edge position to be indicated to sub pixel accuracy. This approach was not suitable for this work as an output image of the same size as the input image was required.

A question which arises in the implementation of the MH operator is what to do about truncation of the mask due to the square shape of the convolution window? This is particularly important when small window sizes are used for large values of sigma. Two approaches were used here;

1. The values within the mask should add to zero if no truncation has taken place. Therefore once a sigma value has been chosen a window size can be calculated to provide a mask truncation error of less than a set value. The value used in this implementation was 0.01.

2. The second approach is to add an offset to all the values within the mask to make it sum to zero.

Marr also suggested that the image should be convolved with a number of masks each using a different value of sigma. The idea was that an edge representing a real object should appear throughout a range of spatial frequencies. This post processing stage which combines the output of a set of MH filters is not used in the work described in this thesis.

Some examples of the effect of using the MH filter, as described, are shown below in Figure 2.3 and Figure 2.4. The first set of images shows the result of changing sigma. These three images are shown with the output grey level scaled by the slope of the zero crossing and using a dead-band around zero of 0.01. The alternative output form, in which the output grey level represents the distance from the nearest zero crossing is shown in Figure 2.4. The effect of using a smaller window size is shown in the second row of Figure 2.4. The smaller window size can be compensated for by introducing an offset to counteract mask truncation (Figure 2.4 D & E). Background edges created by noise can be removed by setting a larger dead-band around zero (see Figure 2.4 C& F). Further examples of the Marr-Hildreth operator are given in Appendix A, including the generation of phantom edges and inherent difficulties with T intersections.

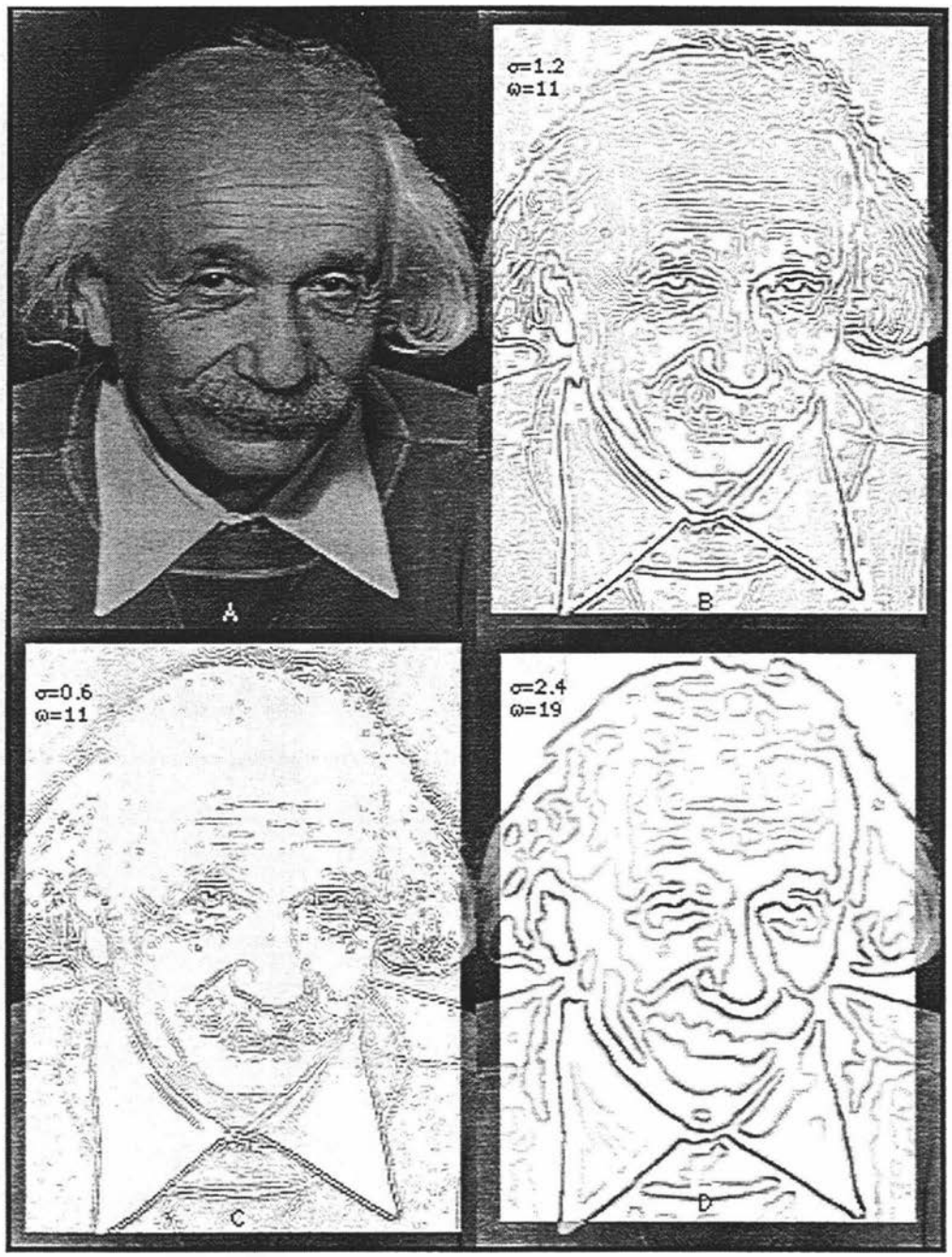


Figure 2.3: Marr-Hildreth Edge Filter

A Original image.	**B** Output from MH operator sigma set to 1.2 window size of 11X11
C Output from MH operator sigma set to 0.6 window size of 11X11	**D** Output from MH operator sigma set to 2.4 window size of 19X19

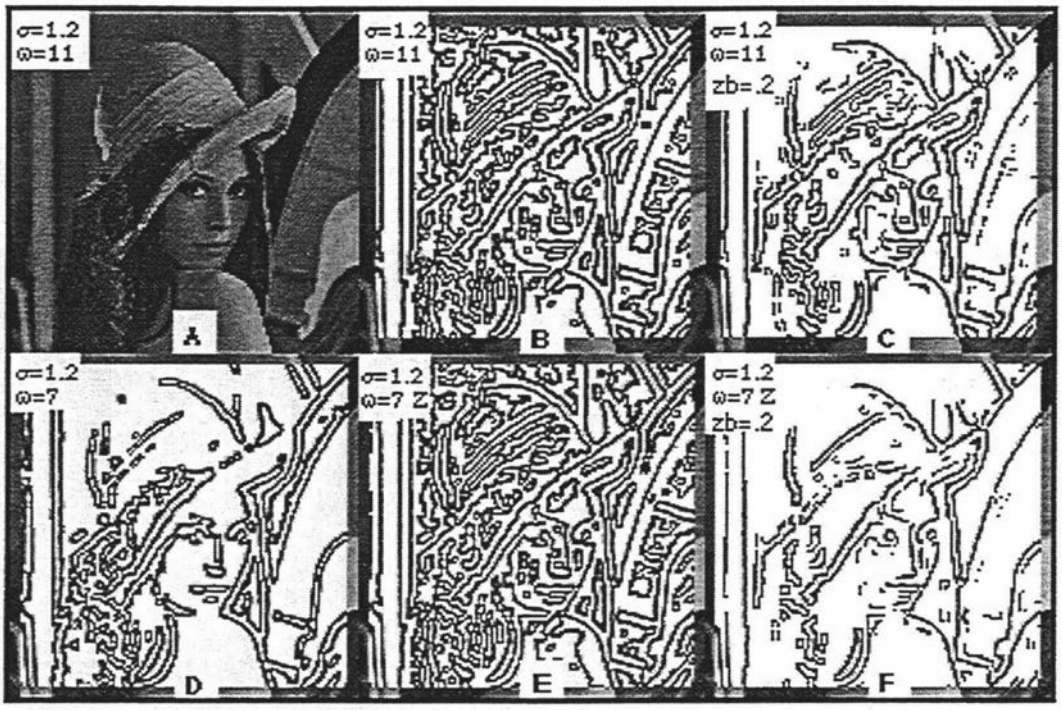


Figure 2.4: Marr-Hildreth Edge Filter Option Effects

Edge pixel grey level represents distance from zero crossing.

A	Original image	**B**	Output of MH with 11x11 window dead band = 0.01 sigma = 1.2	**C**	Output of MH with 11x11 window dead band = 0.2 sigma = 1.2
D	Output of MH with 7x7 window dead band = 0.01 sigma = 1.2	**E**	Output of MH with 7x7 window mask offset used dead band = 0.01 sigma = 1.2	**F**	Output of MH with 7x7 window using mask used dead band = 0.2 sigma = 1.2

2.2 Neural Networks and Back-Propagation

The field of artificial neural networks (ANNs) involves the use and study of a range of models and techniques which are either based on or inspired by knowledge of biological neural systems. Biological systems use highly interconnected networks of relatively simple parallel processing elements to control interaction with the environment and produce "intelligent" behaviour. The study and use of the emergent properties of such processing systems is what defines the field of artificial neural networks.

The field is unusual in that it draws together people from a wide range of backgrounds; engineering, psychology, neuro-physiology, computer science, mathematics, physics and many others. One of the major impacts of the field has been to bring these different specialists together, many of the important insights have come from the resulting cross fertilisation of ideas. The way in which the field is introduced often reflects the background and interests of the particular person introducing it. To some researchers the biological plausibility of the networks is of paramount importance, for some, NNs merely represent an opportunity to re-apply techniques which have been successful in their own specialist field, for some NNs represent another tool for solving difficult problems, for others it provides a method of understanding the behaviour of biological systems, or even ourselves, a little better.

The viewpoint taken here is largely an engineering one, although biased towards looking for simple biological analogies wherever possible. The neural network paradigms are viewed as alternative problem solving techniques which can be evaluated for a variety of real world problems. The bias towards the use of biological analogies results from the observation that the process of evolution leads to the discovery of clever ways of "cheating", that is of extracting useful information from data by methods which may not be exact but which are simple and sufficient. For example finding motion boundaries without measuring optical flow [Srinivasan & Sobey 1992]. It is efficient in engineering terms to, wherever possible, make use of these methods in constructing artificial systems.

Neural network models involve four aspects which are described below (see also Figure 2.5 on page 2.13).

- **A model for individual neurons.** This describes the basic structure of the individual processing units. The most common is a unit which multiplies each input to the neuron by a weight, sums the resulting products and then passes the result through a squashing function.

- **A topology or pattern of connectivity.** This describes how many neurons there are, how they are arranged and what connections can, at least potentially, exist between the individual neurons.

- **A learning rule.** This describes the way in which connection strengths will be varied as a result of experience. For some networks the connection weights may be fixed, for others parameters in addition to connection weights will be varied as a result of experience.

- **A network context.** This describes; what the network is to be used for, what inputs to the network there are, what they represent, what outputs there will be and what they represent.

One classification system for ANNs is based on the way in which experience is made to effect the network. This results in three categories; fixed networks, supervised networks and unsupervised networks. In fixed networks the connection strengths are pre-calculated and then fixed. Hopfield networks [Hopfield & Tank 1986] which settle to a solution are an example of this type. Supervised networks have variable weights and are trained by the presentation of a series of input and target patterns, Perceptrons were the first version of this style of network. A sub-class of supervised networks contains those in which reinforcement learning is used. In this case performance feedback is less frequent and does not provide information on any individual input-output pattern pair. Unsupervised networks have modifiable connection weights but are trained by the presentation of a series of input patterns. The weights change as a result of the nature of the data presented. Kohonens self organising maps are an example of this style of network [Kohonen 1984].

ANNs can be implemented in a variety of ways; simulations on standard computer systems, simulations on a variety of specialist systems or as custom built specialist hardware. Carver Mead's artificial retina [Mead 1989] is an example of the latter.

As suggested above, ANN networks may be constructed for a variety of purposes. The major uses for artificial neural networks are;

- Pattern classification. This involves the forming of decision boundaries to enable a series of input patterns to be classified into two or more groups. Examples include the recognition of handwritten characters, detecting bombs in suitcases and sorting of loan requests [Fukushima 1988].

- Optimisation. Examples include; circuit board routing, solution of the traveling salesman problem, airline flight timetabling [Josin 1987].

- Biological modeling. The use of ANNs to model biological processes and compare theories. Examples include understanding motion detection in insects and modeling of electrosensory systems in sharks [Horridge & Marcelja 1992, Paulin & Nelson 1993].

- Associative memory. Memory systems in which the content and storage location are associated, for fast extraction of related information or completion of partial or noisy data. Examples include the completion of partial or distorted images of faces, the completion of time sequences and retrieval of information from partial probes [Kohonen 1984 ibid., Hinton & Anderson 1989].

- Control & Modeling. The direct use of ANNs as control systems or as process models to allow the use of control techniques which make use of a process model. Key advantages are the ability to model non-linear processes. Examples include control of robotic arms, autonomous control of vehicles and control of chemical processes and plants. [Poo et.al. 1992, Grandvalet et al. 1991].

- Arbitrary function approximation. As networks have been shown to be able to map any well behaved arbitrary function ANNs can be used as an alternative to polynomial and other approximation techniques [Hornik et al. 1989, Cardaliaguet & Euvrad 1992].

The field of neural networks in many ways began with a suggestion by Hebb in 1949 that learning in humans might be the result of a modification in the connection strength between neurons [Hebb 1949]. The Perceptron which was enthusiastically promoted by Rossenblat in 1962 represented the next major step in the development of the field [Rosenblatt 1962]. He clearly expressed the possible advantages of a neurally inspired approach to computation and developed the Perceptron convergence procedure. Minsky and Papert outlined the severe limitations of single layer Perceptrons in the book "Perceptrons An

Introduction to Computational Geometry" [Minsky & Papert 1969]. For a period following the publication of this work relatively little neural network research was done. A full discussion of the early development of the field is provided in many texts [Rumelhart & McCleland 1986, Widrow & Lehr 1990 or Anderson & Rosenfeld 1988].

The important step of generalising the Perceptron learning rule to cover multi-layer systems was finally made by a number of different people. [Werbos 1974, Parker 1982, Rumelhart et al. 1986]. It was no doubt delayed by the suggestion that "*There is no reason to suppose that any of these virtues carry over to the many layered version. Nevertheless, we consider it to be an important research problem to elucidate (or reject) our intuitive judgment that the extension is sterile.*" [Minsky & Papert 1969 ibid.]. The generalisation to multi-layer systems provides the basis for the multi-layer back propagation (BP) network which has been used in the neural network based window filter described in chapter three. The neuron model and topology for such a network is shown below in Figure 2.5.

2.2.1 Back Propagation

Back propagation enables the use of multi-layer networks with supervised learning, by propagating the errors back through the network thus providing information for modification of weights in the hidden layers [Rumelhart et al. 1986]. It is essentially a steepest descent minimisation technique and can therefore suffer from local minima problems [Press et al. 1988, Hausner 1971]. Although there are definitely situations in which this happens a surprising aspect of neural network research is the number of situations in which local minima do not cause problems.

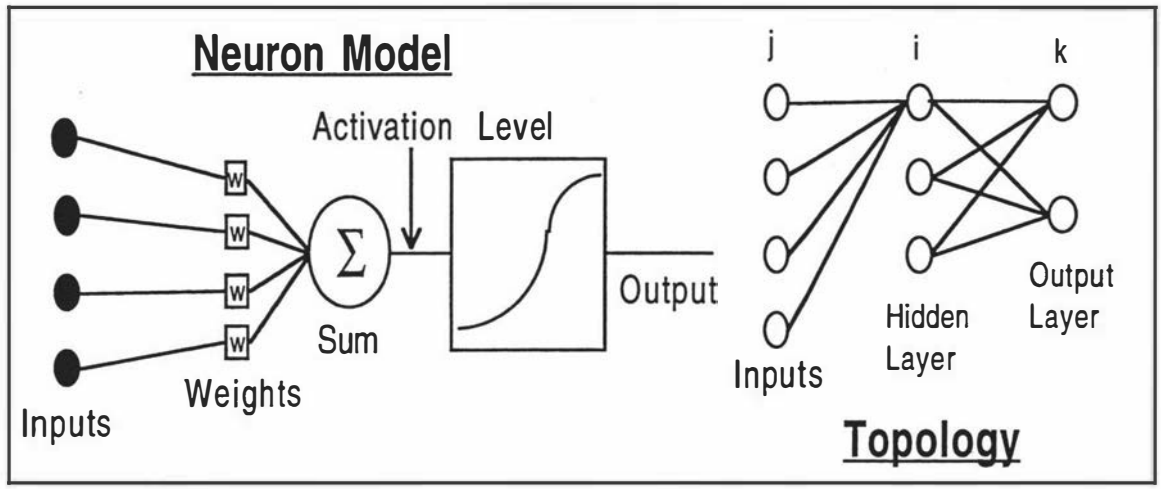


Figure 2.5: Back propagation network

The application of the BP algorithm to each training pair, involves two phases: in the first an input pattern is presented and signals are propagated forward through the network to compute the output for each neuron. For the units in the output layer the output is compared with the target value and an error is computed. In the second phase a backward pass through the network occurs during which a delta term is computed for each unit. Then the weight modification for each connection is calculated making use of these delta terms.

For our network using the subscripts shown in Figure 2.5 and a logistic activation function we have.

Activation for unit i $$a_i = \sum_j w_{ij} o_j + b_i \quad \text{.....2.1}$$

and output for unit i $$o_i = f(a_i) = \frac{1}{1+e^{a_i}} \quad \text{.....2.2}$$

$\Rightarrow$ (note the simple derivative) $$f'(a_i) = \frac{d f(a_i)}{d a_i} = o_i(1-o_i) \quad \text{.....2.3}$$

weight change for connection from unit j to unit i

$$\Delta w_{ij}(t) = -G\frac{\partial E}{\partial w_{ij}} + M\Delta w_{ij}(t-1) \qquad \textbf{2.4}$$

$$= G\delta_i o_j + M\Delta w_{ij}(t-1) \quad \text{.....}$$

where G is the gain, M is the momentum, and δ is defined below
Suffix (t) indicates value at time t

and the error measure E is defined by $E = \frac{1}{2}\sum_k (t_k - o_k)^2$ **2.5**

where t_k is the target for output unit k

The δ signal is defined recursively as follows.

For an output unit k $$\delta_k = (t_k - o_k) f'(a_k) = (t_k - o_k) o_k (1 - o_k) \quad \text{.....2.6}$$

and for hidden unit i $$\delta_i = \left(\sum_k w_{ki} \delta_k \right) f'(a_i) = \left(\sum_k w_{ki} \delta_k \right) o_i (1 - o_i) \quad \text{...2.7}$$

This provides a description of how weight changes can be calculated by use of a series of delta terms which are calculated recursively starting at the outputs of the network. It is also worth considering the derivation of these equations, as modifications to standard BP discussed in chapter three will refer back to aspects of this derivation.

2.2.2 Derivation of Back Propagation Equations

As noted back propagation is essentially a gradient descent algorithm. Gradient descent is based on the idea that taking small steps in a down hill direction from any starting point will take us closer to a minimum of the function. Thus for a function F of x the direction of the change of x is given by $\nabla x = -\frac{dF}{dx}$. The minimum which is approached may be either a local or a global minima [see §3.4.3 for further discussion].

Gradient descent merely involves moving a small distance in the appropriate direction. Two other factors need to be determined; how small is small and when to stop. The derivative term only gives the sign of the desired movement. The distance to move requires further elaboration. A convenient computational approach is to move a distance given by $-G\frac{dF}{dx}$, where G is a suitable constant. This has the added advantage that at points near a minima $\frac{dF}{dx}$ will be small so that steps of smaller and smaller length will be taken as the minimum is approached. The size of change in F can then be used as the criterion for termination. $\frac{dF}{dx}$ may also be small at points other than maxima or minima and this can lead to slow convergence [see §3.4.3]. This still leaves a parameter G which must be chosen to be large enough to provide fast convergence but small enough to avoid oscillation. This is discussed further in chapter three [see §3.4.3]

If the function of interest is a function of more than one variable then a series of partial derivatives can be taken. The direction of required movement in each dimension is given by the negative of the partial derivative. For steepest descent, which requires a movement in the direction of greatest slope, it is convenient to take a step given by the vector consisting of the set of partial derivatives

$\nabla x = G\left[\frac{\partial F}{\partial x_1}, \frac{\partial F}{\partial x_2}, \ldots\right]$ The effect of using this choice of step direction and step size is discussed further in chapter three [see §3.4.3].

To apply the steepest descent to the modifications of weights in a feedforward network an error measure, which is to be minimised, must be chosen. The partial derivatives of this error function with respect to each of the weights must then be calculated. The standard error function chosen is that given by equation 2.5 above. If a subscript p is used to represent the error on pattern p then the total error to be minimised is E with

$$E = \sum_p E_p \qquad E_p = \frac{1}{2}\sum_k \left(t_{kp} - o_{kp}\right)^2 \quad \text{.....2.7}$$

$$\textit{Note} \quad E > 0 \quad \forall\left(t_{kp} - o_{kp}\right) \neq 0 \quad \text{.....2.8}$$

$$E = 0 \quad \forall\left(t_{kp} - o_{kp}\right) = 0 \quad \text{.....2.9}$$

Before proceeding to the calculation of $\frac{\partial E_p}{\partial w_{ij}}$ a justification in terms of stability is required for setting the weight change for w_{ij} $\Delta w_{ij} = \frac{\partial w_{ij}}{dt} = -G\frac{\partial E}{\partial w_{ij}}$ 2.10

Now $$\frac{dE}{dt} = \sum_k \frac{\partial E}{\partial o_k}\frac{do_k}{dt} = \sum_k \frac{\partial E}{\partial o_k}\frac{do_k}{\partial w_{ij}}\frac{\partial w_{ij}}{dt} \quad \text{.....2.11}$$

the last term being free we can choose

$$\frac{\partial w_{ij}}{dt} = -G\frac{\partial E}{\partial w_{ij}} = -G\frac{\partial E}{\partial o_k}\frac{\partial o_k}{\partial w_{ij}} \qquad G \geq 0 \quad \text{.....2.12}$$

$$\therefore \frac{dE}{dt} = \sum_k \frac{\partial E}{\partial o_k}\frac{do_k}{\partial w_{ij}} \times -G\frac{\partial E}{\partial o_k}\frac{\partial o_k}{\partial w_{ij}} \quad \text{.....2.13}$$

$$= -G\sum_k \left(\frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial w_{ij}} \right)^2 \quad \text{2.14}$$

Thus $$\frac{dE}{dt} < 0 \quad \forall \frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial w_{ij}} \neq 0 \quad \text{2.15}$$

$$= 0 \quad if \ \frac{\partial E}{\partial o_k} \text{ or } \frac{\partial o_k}{\partial w_{ij}} = 0 \quad \forall k \quad \text{2.16}$$

These together with 2.9 and 2.10 are the Lyapunov requirements for stability and are conservative. $\therefore \frac{\partial w_{ij}}{dt} = -G\frac{\partial E}{\partial w_{ij}}$ will ensure a stable algorithm.

Using the chain rule we have

$$\frac{\partial E_p}{\partial w_{ij}} = \frac{\partial E_p}{\partial a_i} \frac{\partial a_i}{\partial w_{ij}} \quad \text{2.17}$$

by equation 2.1 we see that the second factor is

$$\frac{\partial a_i}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \sum_j w_{ij} o_j + b_i = o_j \quad \text{2.18}$$

and if we use the symbol δ to represent the negative of the first term in the RHS of 2.17 we have

$$\frac{\partial E_p}{\partial w_{ij}} = -\delta_{pi} o_j \quad with \quad \delta_{pi} = -\frac{\partial E_p}{\partial a_i} \quad \text{2.19}$$

and hence $$\Delta w_{ij} = -G\frac{\partial E}{\partial w_{ij}} = \delta_i o_j$$

This provides the derivation of equation 2.4 above (ignoring the momentum term). It is now necessary to derive the two forms of the equation for δ for hidden units and output units. Again we can use the chain rule to write

$$\delta_{pi} = -\frac{\partial E_p}{\partial a_i} = -\frac{\partial E_p}{\partial o_i} \frac{\partial o_i}{\partial a_i} \quad \text{2.20}$$

using equation 2.2 we can calculate the second term above as

$$\frac{\partial o_i}{\partial a_i} = f'(a_i) = o_i(1 - o_i) \text{ using 2.3} \quad \text{2.21}$$

For the first term of the RHS of equation 2.20 above we have two cases depending on whether the unit in question is an output or a hidden neuron. For an output unit using equation 2.8 above we have

$$\frac{\partial E_p}{\partial o_k} = \frac{\partial}{\partial o_k}\left(\frac{1}{2}\sum_k (t_{kp} - o_{kp})\right) = \frac{1}{2}(2)(t_{pk} - o_{pk})(-1) = -(t_{pk} - o_{pk}) \quad \text{.....2.22}$$

Thus for an output unit we have $\delta_{pi} = (t_{pk} - o_{pk})o_i(1 - o_i)$ which is equation 2.6

For any unit which is not an output unit we can write the first term of the RHS of equation 2.20 as

$$\frac{\partial E_p}{\partial o_i} = \sum_k \frac{\partial E_p}{\partial a_k}\frac{\partial a_k}{\partial o_i} \quad \text{using 2.1 gives} = \sum_k \frac{\partial E_p}{\partial a_k}\frac{\partial}{\partial o_i}\left(\sum_j w_{ij}o_j + b_i\right) = \sum_k \frac{\partial E_p}{\partial a_k}w_{ij} \quad \text{....2.23}$$

which substituting for the first term of the RHS using 2.19 gives

$\frac{\partial E_p}{\partial o_i} = \sum_k -\delta_{pk}w_{ij}$ using this and equation 2.21 to substitute in 2.20 gives

$$\delta_{pi} = -\frac{\partial E_p}{\partial o_i}\frac{\partial o_i}{\partial a_i} = -\sum_k -\delta_{pk}w_{ij}f'(a_i) = \left(\sum_k \delta_{pk}w_{ij}\right)o_i(1 - o_i) \text{ which is equation 2.7}$$

thus the δ term in the RHS of equation 2.7 always refers to the layer above and hence provides a recursive method of calculating the required values starting at the output layer and working towards the input layer.

Finally equation 2.4 also includes a momentum term which modifies the weight changes produced by standard BP to include a fraction of the weight change from the previous iteration. This is done to provide a form of momentum which allows larger values of gain(G) to be used without oscillation. The effect of the momentum term is to filter out high frequency changes.

A final consideration is the use of pattern or epoch weight updates. In the derivation above, the error to be minimised is the error on the entire training set or epoch. Gradient descent therefore requires that the weight changes should be accumulated for a complete epoch and then used to produce a resultant change in each weight. This is referred to as epoch based weight update.

To allow weights to change more quickly and hence produce faster convergence a modification to this procedure is often made. If the weight changes calculated are actually applied just before the presentation of each new

pattern then this is referred to as pattern based weight update. This is usually used at least in the initial stages of training.

2.3 INSTRUCTIONAL DESIGN

The application of Instructional Design (ID) to the training of artificial neural networks is considered in detail in chapters six and seven. The field of ID is briefly introduced here to provide a background for the discussion in the intervening chapters.

Instructional Design or "Design Science" is that part of the general field of education which is concerned with prescribing optimal methods of instruction to bring about desired changes in knowledge and skills for a given set of students [Reigeluth 1983a]. It is clearly related to learning theory but has a different focus. Instructional design focuses on the entire environment in which the pupil is placed and how this will effect learning. This is not to deny the importance of the learner but merely to focus on creating the best learning environment and to look at the interaction between a student (or network) and a training environment from this point of view. Briefly ID deals with how best to teach, or present, a given course to given students and tries to provide a blueprint of what methods to use for particular desired outcomes and students.

There are three aspects of the interaction between teacher, training environment and student which should be considered in instructional design. These are;

- Methods - Manipulation of relevant variables which can be controlled.
- Conditions - Consideration of relevant variables which are fixed but will effect the process of learning.
- Outcomes - The desired changes which are hoped for and should be produced by applying particular methods in particular situations.

Methods can be further subdivided into organisational strategies, delivery strategies and management strategies. In proposing a theory of instructional design an attempt is being made to provide a complete set of strategy components and guiding principles to suggest which strategy components will be best matched to the achievement of particular desired outcomes and conditions. In chapter six several instructional design theories are critically examined and their relevance to training artificial neural networks is discussed.

Naturally there will be differences between ID for people and ID for artificial neural networks, and these are considered in detail in chapter six, however it is suggested that the analogy to teaching people is a useful one. At least some of the techniques which make learning easier for people will also provide benefits if applied to the training of ANNs.

Instructional design has not previously been related to neural network research although interest is developing in a number of techniques which are related to instructional design, for example; learning from hints [Suddarth & Holden 1991, Abu-Mostafa 1993] and structuring and generation of examples [Shirvaikar & Trivedi 1995, Hrycej 1992].

2.4 DISCUSSION

This chapter has briefly introduced; window filters, neural networks and instructional design. The question arises as to why these three areas have been combined. Window filters were chosen as a vehicle for research on the use of neural networks in image processing and for research on methods of improving the performance of these networks used because;

- Window filters are a well understood and heavily used part of the image processors toolbox. This provides firm foundations for the new work and implies that if a general neural network based window filter can be constructed it is likely to be of use. The range of operations which can be performed by window filters has been extensively studied. This should provide opportunity for comparison between the NNWF performance and the performance of other conventional window filters.

- A large training set can be provided with relative ease. This avoids one of the difficulties often encountered in neural network research, that of finding training data sets of adequate size. A single pair of images, one as input and the second as a target, provides one training pattern for the neural network window filter at each pixel position. Thus a single pair of 256x256 images will provide 64,516 training pairs.

- The use of window filters results in relatively small networks whose operation can be simulated on standard computers in a reasonable time frame. This allows a variety of techniques to be investigated.

- An advantage of using a neural network for image processing is that the required task can be defined by example, this eliminates the need to generate an explicit algorithm for each new task. For example, while window filters are extensively used in image processing the choice of filters for particular tasks is a heuristic process which relies heavily on the experience and skill of the image processor. For this reason an alternative approach in which a universal window filter learns the required function from examples should be of general interest. This avoids the need to express mathematically the

relationship between the required action and the parameters and filter needed to implement it. Examples can be produced either by hand by a specialist in the particular application area or by an image processing expert who processes some representative images by using a series of standard tools. This may allow non image processing specialists to obtain useful results or allow large sets of images to be processed at higher speed by a hardware implementation of the NNWF.

Instructional design and the use of an educational environment are investigated as a means of improving both the performance and the applicability of the neural network based window filter. Instructional design has been introduced only briefly in this chapter. Chapter six which is devoted to the examination of theories of instructional design and their relevance to the training of artificial neural networks, provides a more complete introduction.

2.5 REFERENCES

page first cited

Abu-Mostafa 1993

Y.S. Abu-Mostafa, *Learning from Hints*, ACNN, Melbourne, Feb. 1993, pp37-40 ..... 2.19

Anderson & Rosenfeld

J.A. Anderson, E. Rosenfeld, Neurocomputing: Foundations of Research, MIT Press, Cambridge Mass., 1988 ..... 2.12

Cardaliaguet & Euvrad 1992

P. Cardaliaguet, G. Euvrad, *Approximation of a function and its derivative with a neural network*, Neural Networks, Vol. 5, 1992, pp207-220 ..... 2.11

Freeman 1986

H. Freeman, *Image Processing and Pattern Recognition: A General Overview.*, Advances in Image Processing and Pattern Recognition, Elsevier Science Publishers, B.V. North-Holland, 1986, pp224-228 ..... 2.2

Grandvalet et al. 1991

Y. Grandvalet, S. Canu, D. Meizel, T. Lanlois, *Preconditioning for improving convergence of back propagation algorithm in control applications*, in Intelligent engineering systems through artificial neural networks: proceedings of the Artificial Neural Networks in Engineering Conference, ASME Press, New York, 1991, pp89-94 ..... 2.11

Hausner 1971

A. Hausner, Analog and Analog/Hybrid Computer Programming, Prentice-Hall, Englewood Cliffs N.J., 1971, Sec. 11.4 ..... 2.12

Hebb 1949

D.O. Hebb, The Organization of Behavior, Wiley, New York, 1949 ..... 2.11

Hinton & Anderson 1989

G. Hinton, J. Anderson, Parallel Models of Associative Memory Updated Edition, Lawrence Erlbaum, New Jersey, 1989 ..... 2.11

Hopfield & Tank 1986

J.J. Hopfield & D.W. Tank, *Simple "Neural" Optimization Networks: An A/D Converter, Signal Decision Circuit, and a Linear Programming Circuit*, IEEE Transactions on Circuits And Systems, Vol CAS-33, No 5, May 1986, pp533-541 ..... 2.10

Hornik et al. 1989

K. Hornik, M. Stinchcombe, H. White, *Multilayer Feedforward Networks are Universal Approximators*, Neural Networks, Vol. 2, No 5, 1989, pp359-366 ..... 2.11

Horridge & Marcelja 1992

G.A. Horridge, L. Marcelja, *On the Existence of "Fast" and "Slow" Directionally Sensitive Motion Detector Neurons in Insects*, Third Australian Conference on Neural Networks, Canberra, Feb. 1992, pp1-3 ..... 2.11

Hrycej 1992

T. Hrycej, *Supporting supervised learning by self-organisation*, Neurocomputing, V4, N1-2, 1992, pp17-30 ..... 2.19

Jain 1989
A.K. Jain, Fundamentals of Digital Image Processing, Prentice & Hall, Englewood Cliffs NJ, 1989, p7..... 2.2

Josin 1987
G. Josin, Neural Network Heuristics, Byte, October 1987, pp183-192..... 2.11

Kohonen 1984
T. Kohonen, Self-Organization and Associative Memory, Third Edition, Springer Verlag, Heidelberg 2.10

Marr & Hildreth 1980
D. Marr & E. Hildreth, *Theory of Edge Detection*, Proceedings of Royal Society London, B 207, 1980, pp187-217..... 2.3

Mead 1989
C.A. Mead, Analog VLSI and Neural Systems, Reading Mass., Addison Wesley, 1989, Ch 15. 2.10

Minsky & Papert 1969
M. Minsky & S. Papert, Perceptrons An Introduction to Computational Geometry, MIT Press, Cambridge Mass., 1969..... 2.12

Overington 1992
Ian Overington, Computer Vision: A unified, biologically-inspired approach, Elsevier, Amsterdam, 1992..... 2.2

Parker 1982
D.B. Parker, *Learning Logic*, Invention Report, S81-64, File 1, Office of Technology Licensing, Stanford University, October 1982..... 2.12

Paulin & Nelson 1993
M.G. Paulin, M.E. Nelson, *Combining Engineering Models with Biophysical Models to Analyze a Biological Neural Network: The Electrosensory System of Sharks, Skates and Rays*, Proceedings of Artificial Neural Networks and Expert Systems Conference, Dunedin, 1993, pp8-10..... 2.11

Poo et al. 1992
A.N. Poo, M.H. Ang Jr, C.L. Teo, Qing Li, *Performance of a neuro-model-based robot controller: adaptability and noise rejection*, in Intelligent systems Engineering, Autumn 1992, pp50-62.... 2.11

Press et al. 1988
W.H. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling, Numerical Recipes in C: The Art of Scientific Computing, Cambridge University Press, Cambridge, 1988..... 2.12

Reigeluth 1983a
C. M. Reigeluth, *Instructional Design: What is it and why is it?*, pp3-36, In Instructional-Design Theories and Models, Lawrence Erlbaum Associates, New Jersey, 1983 ..... ~~2.18~~ 2.19

Rosenblatt 1962
F. Rosenblatt, Principles of Neurodynamics, Spartan, New York, 1962..... 2.11

Rumelhart 1986
D.E. Rumelhart, J.T. McCleland, Parallel Distributed Processing, Explorations in the Microstructure of Cognition. Three Volume Set, MIT Press, Cambridge MA, 1986, pp41-44..... 2.12

Rumelhart et al. 1986
D.E. Rumelhart, G.E. Hinton, R.J. Williams, *Learning internal representations by back-propagating errors*, Nature, 323, 1986, pp 533-536..... 2.12

Russ 1992
J C. Russ, The Image Processing Handbook, CRC Press, North Carolina, 1992, p120..... 2.3

Schwartz 1977

E.L. Schwartz, *Spatial Mapping in the Primate Sensory Projection Analytic Structure and Relevance to Perception*, Bilogical Cybernetics, V25, pp181-194..... 2.2

Shirvaikar & Trivedi 1995

M.V. Shirvaikar, M. M. Trivedi, *A Neural Network Filter to Detect Small Targets in High Clutter Backgrounds*, IEEE Transactions on Neural Networks, Vol. 6 No. 1, Jan 1995, pp252-257..... 2.19

Srinivasan & Sobey 1992

M.V. Srinivasan, P. Sobey, *Neural Networks for the Detection of Motion Boundaries*, in Proceedings of Third Australian Conference on Neural Networks ACNN'92, Canberra, Feb 1992, pp1-3..... 2.9

Suddarth & Holden 1991

S.C. Suddarth, A.D. Holden, *Symbolic-neural systems and the use of hints for developing complex systems.*, International Journal of Man Machine Studies, Vol. 353, Sept. 1991, pp291-311 ..... 2.19

Werbos 1994 1974

P.J. Werbos, *Beyond regression: New Tools for prediction and analysis in the behavioural sciences*, Doctoral Dissertation, Appl. Math., Harvard University, November 1994..... 2.12

Widrow & Lehr 1990

B. Widrow, M.A. Lehr, *30 Years of Adaptive Neural Networks: Perceptron, Madeline, and Backpropagation*, Proceedings of the IEEE, Vol. 78 No. 9, Sept. 1990, pp1415-1442..... 2.12

Chapter 3

3. A UNIVERSAL, NEURAL NETWORK BASED, WINDOW FILTER

Section one of this chapter contains a description of the structure and implementation of a neural network based window filter(NNWF). In section two three experiments are described in which the network filter is trained to emulate standard window filters. These experiments use input-target image pairs to train the network filter. The target image contains the desired transformation, or filtered form, of the input image. These experiments show that a filtering operation can be successfully defined by such an input-target pair of images. The network filter is also shown to be able to generalise the required filtering operation to other images. In section three some additional experiments with the NNWF are described. In section four methods of improving the performance of the network filter are discussed. The final section of the chapter contains a discussion of the results from these initial experiments and consideration of what extensions would be required to provide a **universal** neural network based window filter.

3.1 Structure & Implementation of the NNWF

The NNWF is similar in operation to the window filters described in chapter two. It takes a single image as an input and using a scanning process it builds up an output image. The filter operates in one of two modes; training mode and run mode. In addition to the normal input and output images associated with a window filter the NNWF requires a target image which is only used during training. Once trained, other images can be presented to the NNWF and it will produce a "filtered" version as output. In the first set of experiments where standard filters were emulated the targets were produced by the normal algorithmic implementation of the filters concerned. In the more general case the targets images would be produced either by a domain expert or by an image processing expert who would apply a sequence of operations to a sample image.

3.1.1 Structure of the NNWF

The structure and operation of the NNWF will be explained with reference to Figure 3.1 below. In this figure an input image which contains grey scale printed text is shown on the left, the neural network is shown in the centre and an output image which is being formed by the filter as it scans the input image is shown on the right. In run mode, the filter is scanned across the input image, in each position it takes inputs from a window in the input image and produces a single output pixel in the output image. It is thus similar in operation to a standard window filter, however, rather than using an algorithm or function to process the pixel values from the input window a neural network is used. The pixel values taken from the input window form the set of inputs to the neural network. The output from the network provides a value for the pixel in the output image in the corresponding position. One scan through all pixel positions in an image is referred to as an epoch and produces a single output image. The output image is thus a "filtered" version of the input image.

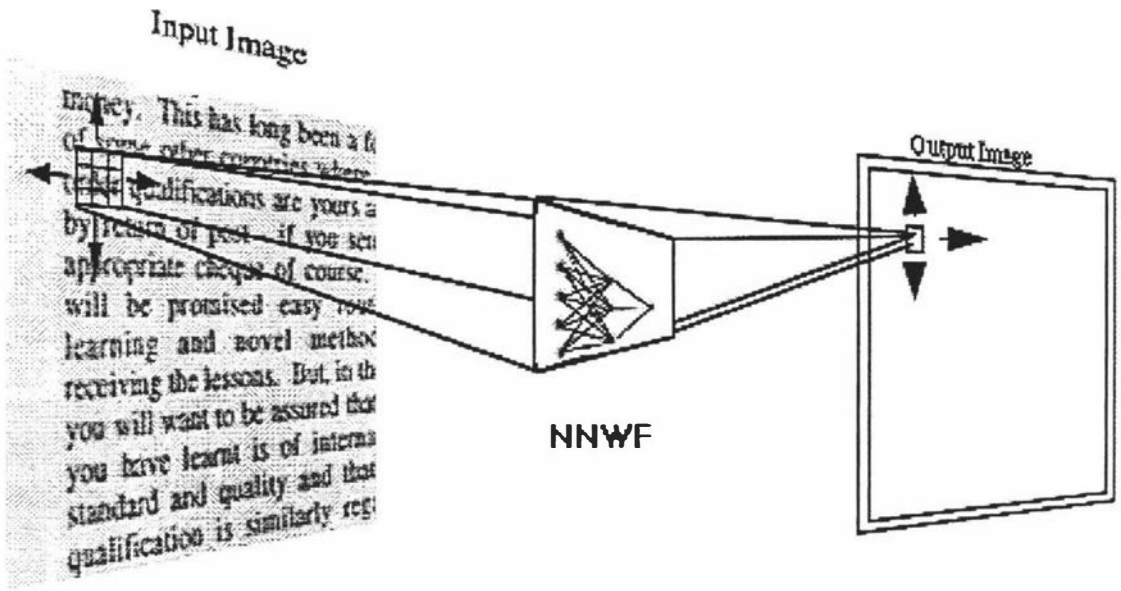


Figure 3.1: Topology of Neural Network Window Filter

The NNWF incorporates a multi-layer neural network with back-propagation learning in which each neuron in the first hidden layer receives weighted inputs from all the pixels in the input window. Before presentation to the network the pixel values are scaled to the normal output range (either 0,1 for neurons with a sigmoid squashing function or -1,+1 for an inverse tangent squashing function). In run mode, each neuron in this first hidden layer performs the following operations;

- Each pixel from the input window is multiplied by a weight.
- The products are summed
- The resulting activation level is scaled by a non-linear squashing function to produce a neuron output.

The outputs from the neurons in this first hidden layer are fed to all the neurons in the next hidden layer. The neurons in the second hidden layer process their inputs in exactly the same way that the scaled pixel values were processed by neurons in the first layer. All of the neuron outputs from the second hidden layer are fed to a single neuron in the output layer. This neuron again performs the same processing as the other neurons in the network. The output of this neuron is then scaled back to the normal (0 - 255) pixel range.

In run mode the weights used to control the connection strengths between pixel values and neurons or between one neuron and another are all fixed. The values

of this set of weights determines the effect of processing an image using the NNWF. The weights are systematically modified during training to make the output produced by the NNWF more similar to the target image. It is hoped that this will produce a filter which will have generalised the process of transforming the input image into the target image. If this is the case then the trained NNWF can be used to process other images in a similar manner.

In training mode the NNWF operates as follows;

> Initially the weights are all set to small random values. The NNWF begins scanning the input window across the input image. At each window position a normal forward pass through the network is performed using the existing weights. This produces an output value for that position. The pixel in the corresponding position in the target image is used to provide a target output value. The difference between these two values defines the error. Using normal back-propagation learning this error signal can be used to generate small changes to all of the network weights. These weight changes can either be applied immediately for pattern weight updates or accumulated during the presentation of a complete image to produce epoch weight updates.

3.1.2 Implementation of the NNWF

The NNWF has been implemented on two separate platforms. The early part of the work described in this thesis was carried out using a Digital Equipment Corporation Vax. The later experiments were carried out using an Apple Macintosh Computer for the image capture and display and a Decstation 3100 running UNIX for the network simulations.

The NNWF was first implemented as a series of extensions to the Vax Image Processing System VIPS [Bailey & Hodgson 1988]. The extensions were written in Vax C. The C code simulated a multi-layer neural network with back-propagation learning and allowed images to be used as network inputs, outputs and training targets. The extensions allowed specification at run time of window size, number of hidden layers, learning rate etc.

Later the NIH Image package running on an Apple Macintosh (Centris 660AV) was used for all the image processing including; image capture, image display

and parameter visualisation. For higher quality image capture a Scion[1] frame grabber card was used. Additions to NIH Image package were written to implement some of the classical filters used in the experiments, described later, and to produce image comparison statistics. These additions were implemented using the scripting facilities of NIH Image, modifications to the NIH Image Pascal source code or NIH Image plug-in filters written in Think C.

Aspirin/MIGRAINES(AM) [Leighton 1991] was used for the neural network simulation running under UNIX. This is a public domain software package for general use in neural network research. It was necessary to provide extensions to AM to allow for the use of images for network inputs, outputs and targets during training. These modifications were written in C. Images and training results are passed between the two systems(Mac and UNIX) automatically using UNIX command scripts which control the UNIX File Transfer Protocol (FTP).

AM compiles network description files(Aspirin files) and produces C code which is linked with user routines and the user interface (MIGRAINES). This produces an executable file which can either be run interactively or controlled from a UNIX command script. Additional utility programs were written to convert the output from the simulations into an image file format which could be read by NIH Image on the Mac. Routines were also added to display the network weights as images and to reformat convergence statistics for graphing and analysis.

The source code for the major additions to the AM package and some examples of the UNIX control files are provided in Appendix B. The source code for the NIH Image additions and Plug-ins is provided in Appendix C. Approximately 1000 lines of code was written during the course of this work.

[1] Scion LG-3, Scion Corporation, Frederick Maryland 21701, USA

3.2 Window Filter Emulation with the NNWF

This initial set of experiments was carried out to answer three questions;

- Can the NNWF perform the filtering operations normally performed by classic window filters?
- Can the training procedure extract the filter characteristics from an example input - target pair of images?
- Can the NNWF learn a generalised filtering operation which will produce the correct results for other input images?

Three filters were emulated; a linear vertical edge detector(LVE), a Sobel filter, and a Marr-Hildreth type filter(MH). These are all described in the previous chapter [§2.1]. They were chosen as representative examples of three different classes of window filters; simple linear filters, non-linear filters and complex rule based filters.

The network used for this series of experiments is also described in the previous chapter [§2.2]. The NNWF window size was chosen to match the mask size used for each of the conventional filters. It was configured with a 3x3 window for the Linear vertical edge and Sobel filter experiments and with a 7x7 window for the MH filter experiment. The MH filter would typically be used with a larger mask size but the 7x7 mask used produced adequate edge information for the test images and reduced processing time for the simulations. A two layer network with nine neurons in the hidden layer and a single neuron in the output layer was used for most of the experiments. Each layer was fully connected to the layer below. Where other configurations were used this is noted in the description of the particular experiment.

A section of the two scenes used in this series of experiments is shown in Figure 3.2 and Figure 3.3 below. The picture of Albert Einstein, which was captured from a calendar print, was chosen as an example of a complex image containing a variety of edges. The microscope image provided an example of a three dimensional scene. These monochrome images were originally captured at a resolution of 256x256 pixels and with an 8 bit grayscale using an Ikegami CCD camera and a Matrox frame grabber attached to the Vax. This image capture system had non square pixels. The later transfer to the Macintosh system with

square pixels resulted in a distorted aspect ratio for some of the images shown in the following sections. The aspect ratio of the images has been corrected for printing except where this produced visible image artifacts.

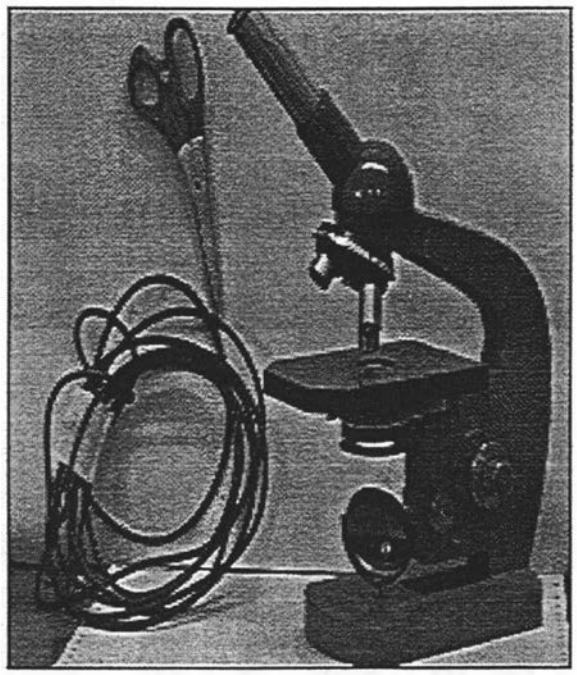


Figure 3.2: Microscope Test Image

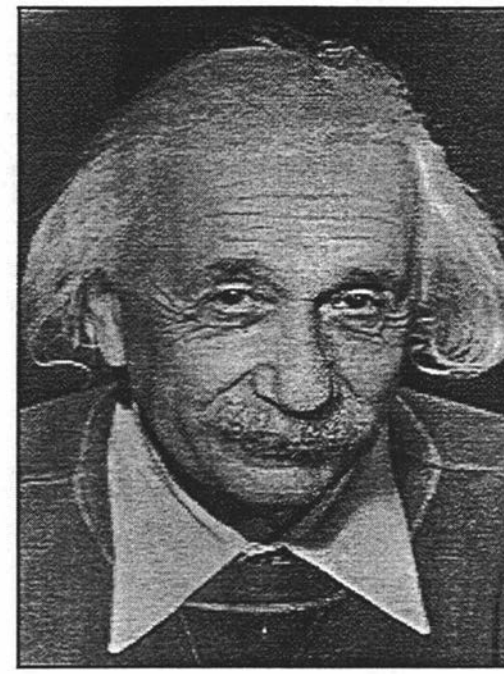


Figure 3.3: Einstein Test Image

To determine the extent to which the NNWF would generalise the required operation from the training set the following procedure was used:

- From each scene a sub-image was extracted and processed using a conventional filter. This provided two training sets; each consisting of a sub-image and the filtered form of the sub-image.
- The NNWF was trained on one sub image training set and then run on both full images.
- The NNWF was then retrained using the other sub image training set and again run on both full images.

This enabled us to compare the performance of the NNWF on a particular image, when trained on a part of that image and when trained on a part of a different image. This technique is related to the technique of cross-validation [Hansen & Salmon 1990].

It is necessary for the training set to provide sufficient variety, in terms of positions in the n-dimensional window input space, to enable the function of the

prototype filter to be determined. For this series of experiments the sub-image size and position was chosen subjectively with these requirements in mind. A small (39x49) sub-image was used for the first two experiments and a larger (128x128) sub-image was used for the Marr-Hildreth emulation experiment. The position and size of these sub-images are indicated by a black square in Figure 3.5, Figure 3.7 and Figure 3.9.

In the results following, one pass in which the window filter is scanned across the training sub-image, is referred to as an "epoch" and represents a number of update cycles for the filter. (For example, the BP algorithm will perform 1739 update cycles per epoch for a 39x49 sub-image, with a 3 x 3 input window and pattern based training)

The peak error and the RMS error were calculated on a pixel by pixel basis as a measure of similarity between images produced by the NNWF and by the classical filters. The peak error is defined as the largest intensity difference between corresponding pixels in two images. For an n by m image pair we have:

$$E_{RMS} = \sqrt{\frac{\sum_{x=1}^{n}\sum_{y=1}^{m}\left(t_{xy} - o_{xy}\right)^2}{n.m}} \quad \text{.....3.1}$$

$$E_{Peak} = \max\left(t_{xy} - o_{xy}\right) \quad x = 1,n \qquad y = 1,m \quad \text{.....3.2}$$

where t is a pixel value in the target image and o is the corresponding pixel value in the output image.

The details of the three filter emulation experiments are discussed in the following sections.

3.2.1 Vertical Edge Filter

A vertical edge filter using the mask shown in Figure 2.2B was used as an example of a simple linear filter. The linear filter performs a convolution with a mask as explained in chapter two [§2.1.1].

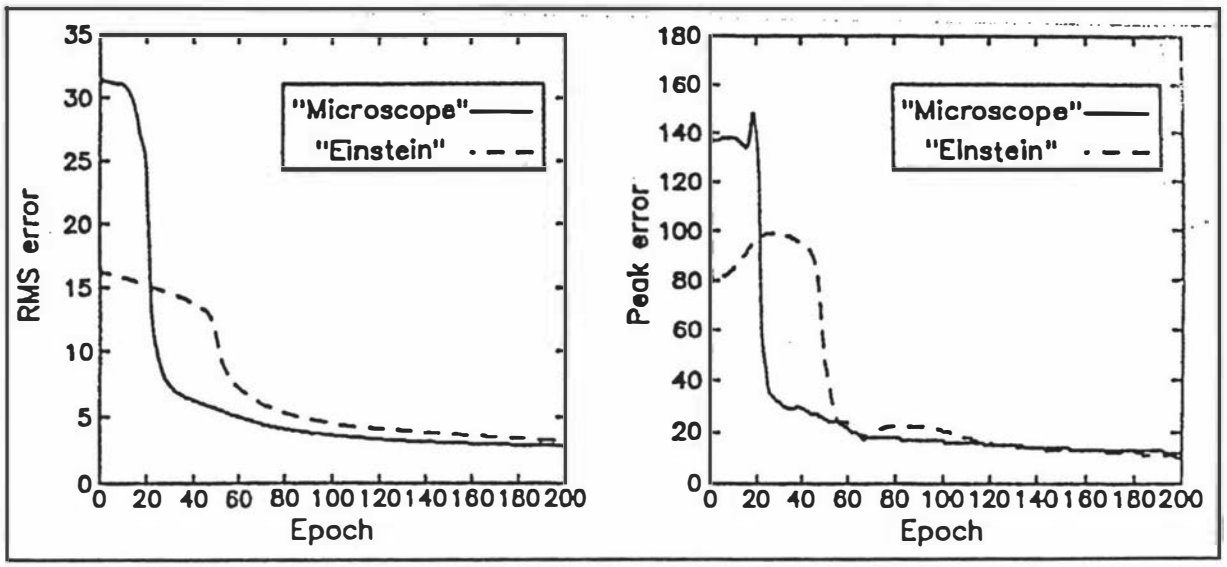


Figure 3.4: Convergence for Linear Filter Emulation

The target output image was created using the VIPS LINEAR FILTER command. For the network used in the NNWF the gain was set to 2 and the momentum was set to 0.5 (See equation 2.4). The network was trained for 200 epochs. The convergence is shown on the graph in Figure 3.4 above. The images produced by the trained NNWF with the microscope scene as input are shown in Figure 3.5. Note this shows both the performance when trained on a sub-image taken from the microscope scene (c) and the performance when trained on a sub-image taken from the alternative Einstein image (d).

Performance on the remainder of the image from which the training sub-image was extracted (see Figure 3.5c) implies that the NNWF is able to perform the required filtering operation. The performance when the training set was taken from a different image (Figure 3.5d) shows that the learnt transform is not dependent on any peculiarities of the particular image used for training and that the NNWF has "generalised" the required transformation.

A similar set of results was obtained when the trained NNWF was applied to the "Einstein" image.

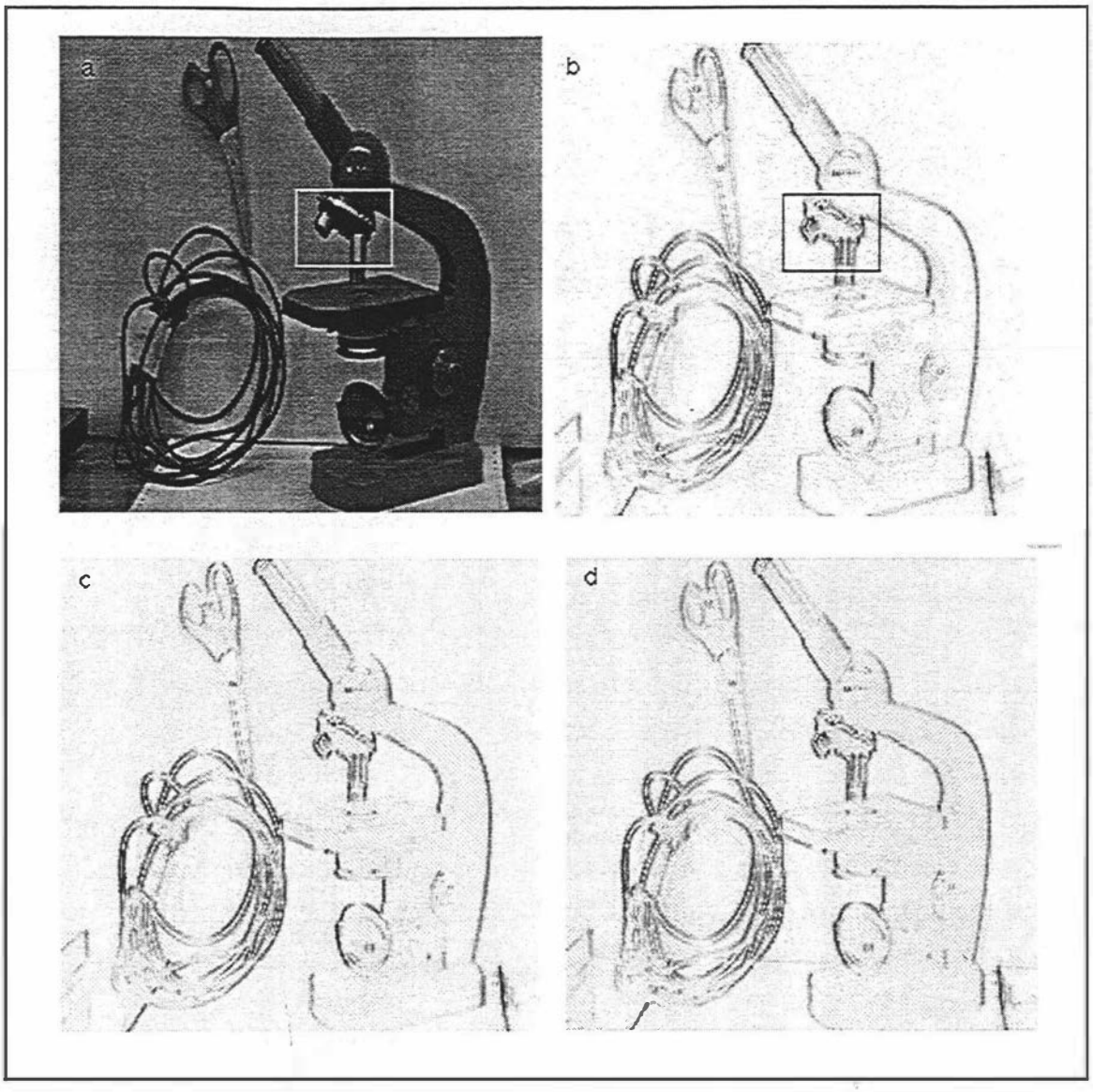


Figure 3.5: Emulation of Linear Vertical Edge Filter

The input image and the output from the vertical edge filter are shown in a and b. The outputs from the NNWF are shown in c and d. Image c was obtained when the NNWF was trained with the sub-images shown in a and b. Image d was obtained when the NNWF was trained with the alternate sub-images shown in Figure 3.7a and b. Comparing these two images (c,d) with the prototype image output (b) gives RMS errors of 3.2 and 5.8 respectively and peak errors of 26 and 53.

3.2.2 Sobel Filter

The second sequence of experiments was based on the Sobel edge detection filter. It is an example of a non-linear filter. Its operation is described in the previous chapter [§2.1.2]. The output of the Sobel filter is the RMS combination of the outputs of two linear filters, one using the specified mask and the other using the transpose of the mask. The mask used in this experiment was:

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

The experimental procedure used was the same as for the linear filter experiments. The required target sub-images were produced by the VIPS FILTER SOBEL command. Two networks were trained each using a sub-image taken from one full image. The error as a function of the number of training epochs, for the two test images, is shown in Figure 3.6 below. The results of applying the trained NNWF to the "Einstein" image are shown in Figure 3.7. Again this shows both the result when trained on a sub-image taken from the Einstein image and the result when trained on a sub-image from the microscope scene.

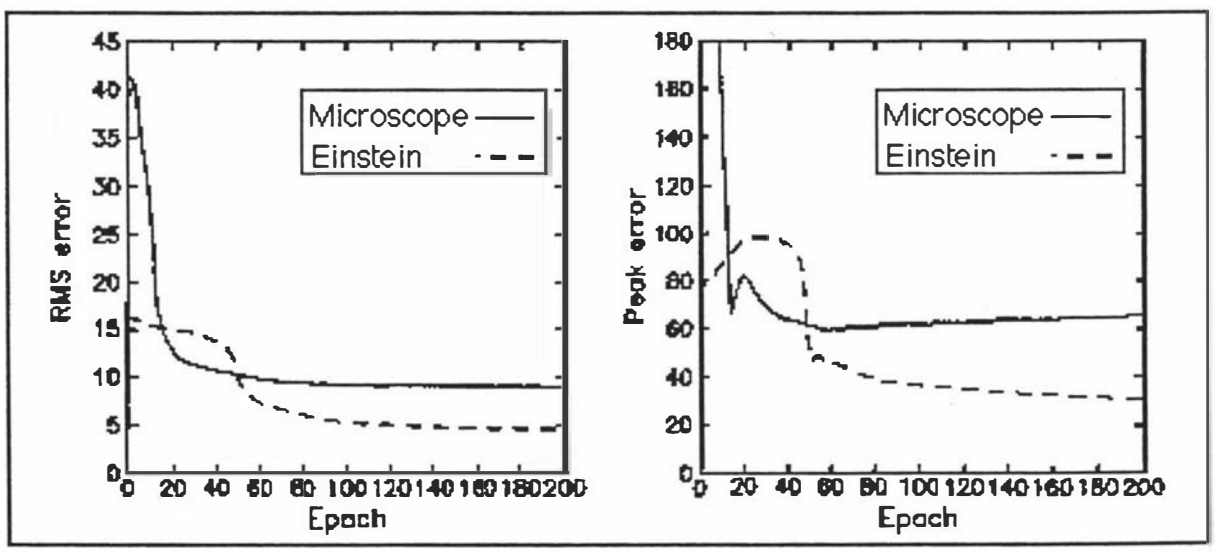


Figure 3.6: Convergence for Sobel Emulation

The same questions can be asked for this sequence of experiments. Again the initial rate of convergence was good and the NNWF was able to generalise the required filtering operation and produce good results on different images.

An interesting observation can be made if the two pairs of sub-images used in training are carefully examined for the presence of edges of various orientations. The sub-images in Figure 3.7 a&b show few vertical edges. The network trained on this pair of sub-images fails to detect the vertical edges of the collar as shown in Figure 3.7c. Whereas the sub-images of Figure 3.5 a&b show very distinct and numerous vertical edges but relatively few and fragmentary horizontal edges, particularly of the type shown in the area of the forehead. The network trained from these sub-images fails to detect horizontal edges in this area as shown in Figure 3.7d. This indicates that both sets of training sub-images are somewhat inadequate epitomes of the required task.

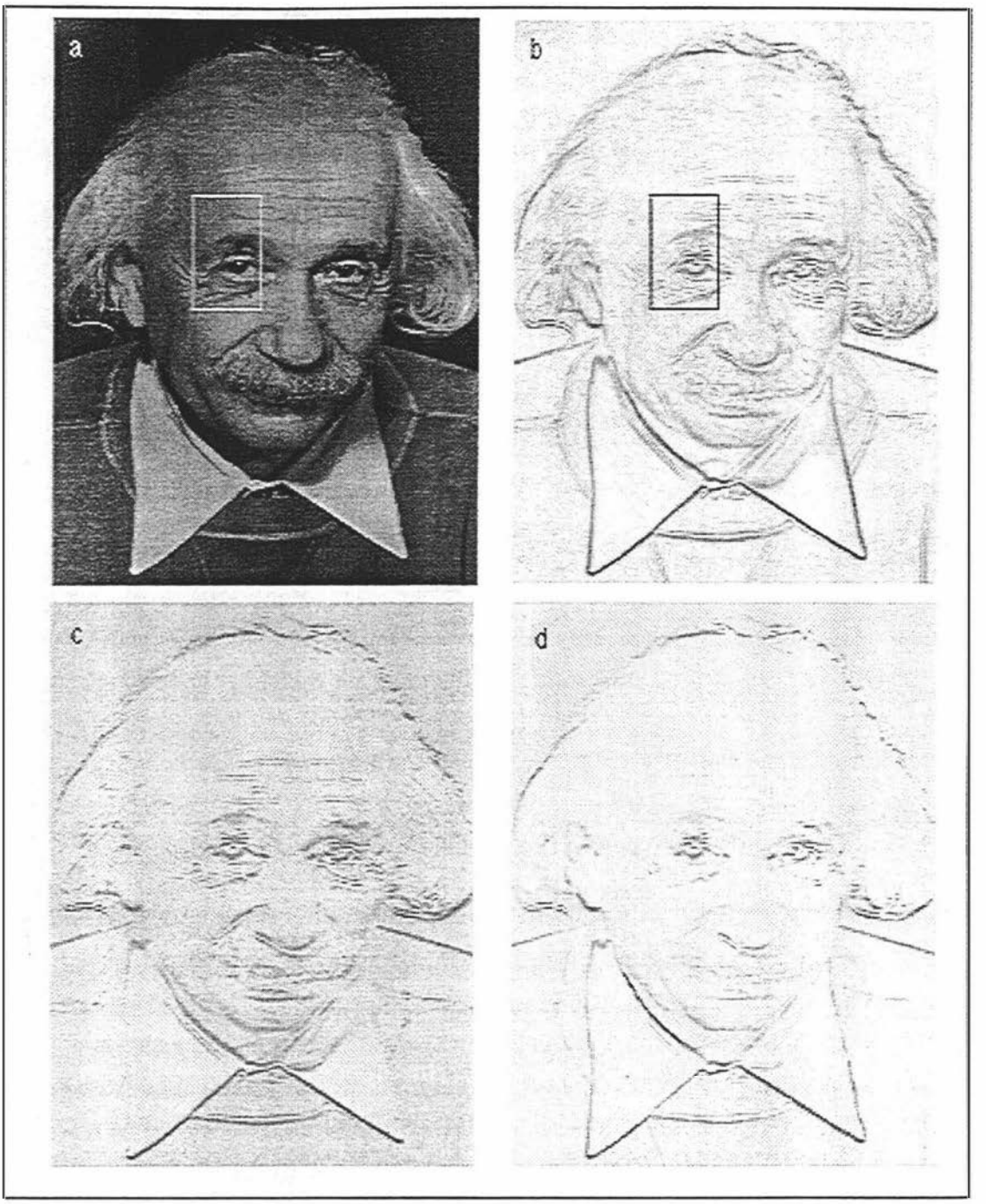


Figure 3.7: Emulation of Sobel Edge Filter

The input image and the output from the Sobel edge filter are shown in a and b. The outputs from the NNWF are shown in c and d. Image c was obtained when the NNWF was trained with the sub-images shown in a and b. Image d was obtained when the NNWF was trained with the alternate sub-image shown in Figure 3.5a. Comparing these two images (c,d) with the prototype image output (b) gives RMS errors of 10.1 and 12.2 respectively and peak errors of 124 and 88.

3.2.3 Marr-Hildreth Filter

The third sequence of experiments was based on a Marr-Hildreth type of edge operator[2]. The Marr-Hildreth filter was used as an example of a complex rule based window filter. Its operation is explained in the previous chapter [§2.1.3]. The particular parameter values chosen for this sequence of experiments were;

Sigma	Window Size	Zero Band	Minimise Sum	Scale by Edge Strength	Threshold Level
1.2	7x7	0.1	off	off	128

It should be noted that the Marr-Hildreth algorithm is a two pass system which first convolves the image with a 7x7 kernel and then compares adjacent pixels to detect zero crossings in the second derivative. This implies information from more than a 7x7 window may be needed to emulate the filter in a single pass, however, the NNWF was initially used with a 7x7 window. In comparison with the experiments on the other filters a larger training sub-image, a smaller gain and a larger number of epochs was used. These changes were necessary as the prototype filter is more complex and operates on a larger window.

The NNWF was trained with target sub-images produced by the MARHIL operator described in the previous chapter. The errors as a function of the number of training epochs are shown in Figure 3.8. Only RMS errors are shown as the concept of peak error is of little use in comparing binary images. It will be zero if the images are identical and 255 in all other cases. The results of processing the two test images with the Marr-Hildreth operator are shown in Figure 3.9a and b. The size of the training sub image is also shown. The results of processing the two test images with the trained NNWF are shown in Figure 3.9c and d. When trained on the sub-image taken from the microscope scene the RMS errors for the two full images were 47.1 and 63.3.

---

[2] These experiment used an earlier implementation of the Marr Hildreth operator which only made comparisons with pixels to the right or below when looking for zero crossings. Later experiments with the newer implementation produced similar results.

The NNWF output images appear quite similar to the prototype outputs even though the RMS errors are still large. Methods of improving this performance are discussed in a later section of the thesis [§5.4.1], however, these error values also indicate a shortcoming in the measures of similarity being used. The RMS error is not particularly suitable for the comparison of binary images so other comparison measures were investigated. Two error measures which have been used for the comparison of edge images by Jain [Jain 1989] are the edge detection error rate "Pe" and a performance measure "Pm" defined as follows:

$$P_e = \frac{n_1}{n_0} \times 100 \quad \textbf{3.3}$$

where n_0 is the number of ideal edge pixels (ie number produced by the prototype filter) and n_1 is the number of missed or new edge pixels

$$P_m = \frac{1}{\max(N_I, N_D)} \sum_{i=1}^{N_D} \frac{1}{1+\alpha d_i^2} \quad \textbf{3.4}$$

where d_i is the distance between a pixel declared as an edge and the nearest ideal edge pixel
α is a calibration constant (set to 1 for our results)
N_I and N_D are the number of ideal and detected edge pixels respectively
As the NNWF produces a graded output for each pixel the output image is thresholded at a level which gives $N_I \approx N_D$ before P_m is calculated.

Both of these measures provide a better match with subjective evaluation than the RMS measure which in some cases varies very little while major subjective changes take place in the output images. For the image in Figure 3.9c the performance measures were Pe = 1.3 and Pm = 0.91.

A number of additional experiments were performed to try to improve the performance of the NNWF on the Marr-Hildreth emulation. These included: Increasing the window size to 9x9 to allow for the two pass nature of the MH algorithm, restricting training to areas surrounding edges, changing the training sub-image (both size and position were varied), updating the weights at the end of each epoch rather than after each pattern, thresholding the NNWF output and using target values of 0.1 and 0.9 rather than 0 and 1.

Only minor improvements were achieved by these means although much better results were obtained later [see §5.4.1]. This improved performance was the result of using one of the training heuristics developed as part of an educational environment introduced in chapter five.

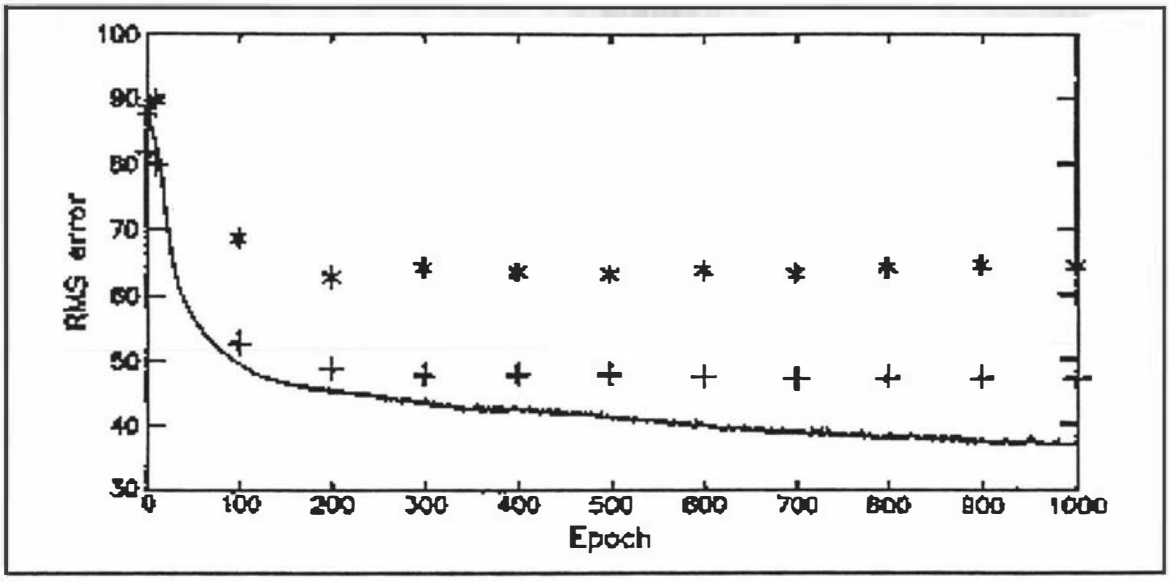


Figure 3.8: Convergence for Marr-Hildreth Emulation

The solid line shows the RMS error for the training sub-image at the end of each training epoch. The output from the partially trained network is compared with the output of the prototype filter for both full images. The + data points are for the image from which the training set was taken. The * data points show the results when the same network filter is used on the alternate image.

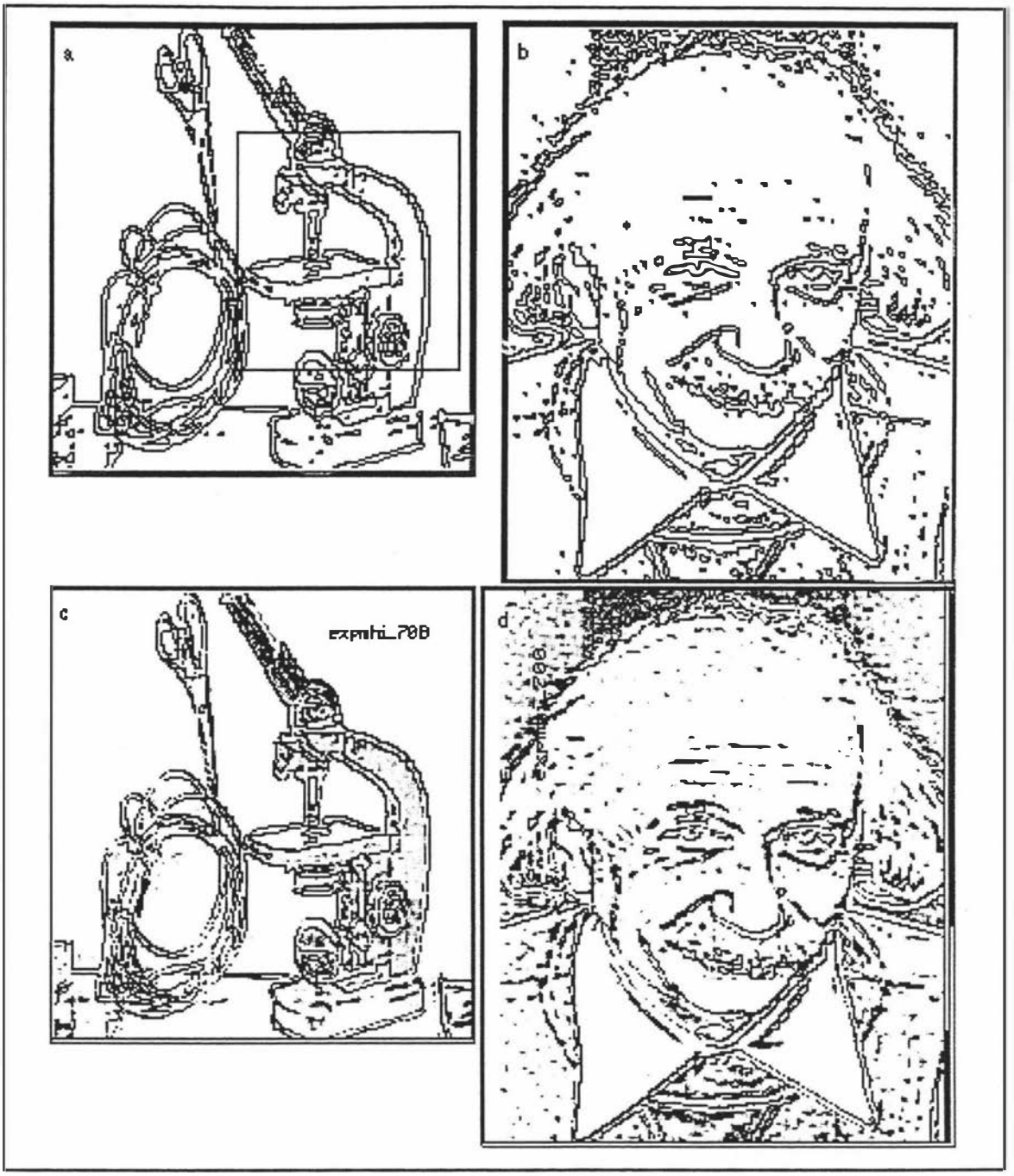


Figure 3.9: Emulation of Marr-Hildreth Operator

The prototype output for this filter can be seen in a and b. The outputs from the NNWF (after training for 700 epochs on the microscope sub-image) are shown in c and d. Comparing the images in a and b with those shown in c and d gives RMS errors of 47.1 and 63.3 respectively.

3.2.4 Effects of Network Parameters

While the initial rate of convergence for both the Sobel and linear filter experiments was good, convergence to an exact match with these "classic" filters was not achieved. The results of ten thousand training epochs for

emulation of the Sobel filter is shown in Figure 3.10, at the end of this run the RMS error was still about 3%. The failure to produce an exact match with classic filter outputs requires further investigation. It may be related to the particular "squashing" or threshold function used in the BP algorithm, the dynamic range of the images being processed or possibly to quantisation or numerical errors. Alternatively, a sub optimal minima may have been reached. As these errors were small in comparison to those in the Marr-Hildreth emulation further investigation of them has been left as future work.

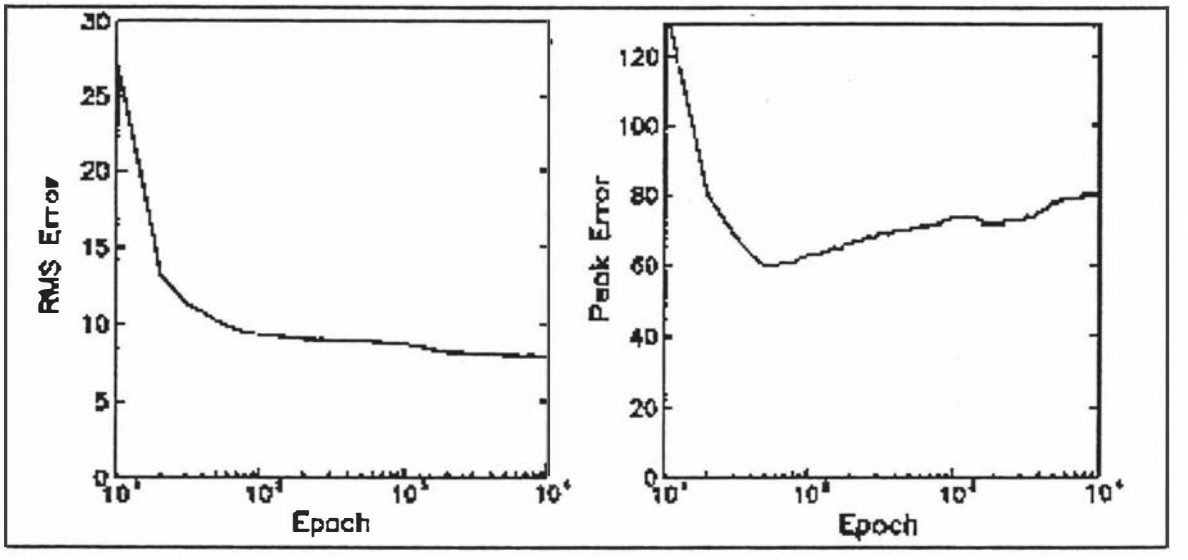


Figure 3.10: Convergence of NNWF for Large number of training epochs

The effects of varying the gain and momentum terms are shown in Figure 3.11. For a fixed momentum value increasing gain improved convergence until a level was found at which the algorithm became unstable. For a fixed value of gain increasing momentum provided faster convergence, however if momentum was set too high then the final error value was increased. These two parameters interact and the optimum values also depend on the size of the input window and the number of inputs connected to each neuron [Tollenaere 1990]. For larger window sizes or high "fan in" smaller values of gain are required. In the literature particular workers tend to choose a value for one of these parameters and then vary the other, a common choice is 0.9 for momentum and values of 0.1 to 0.01 for gain. A more complete analysis of the effects of network parameters requires sufficient computing power to run a large number of simulations in a reasonable time, several studies of this form have been done but generally only for particular networks and particular problem types [Tollenaere 1990 ibid.]. Methods of reducing the number of ad hoc parameters to be chosen are discussed in a later section of the chapter which considers modification to the back-propagation algorithm [§3.4.3].

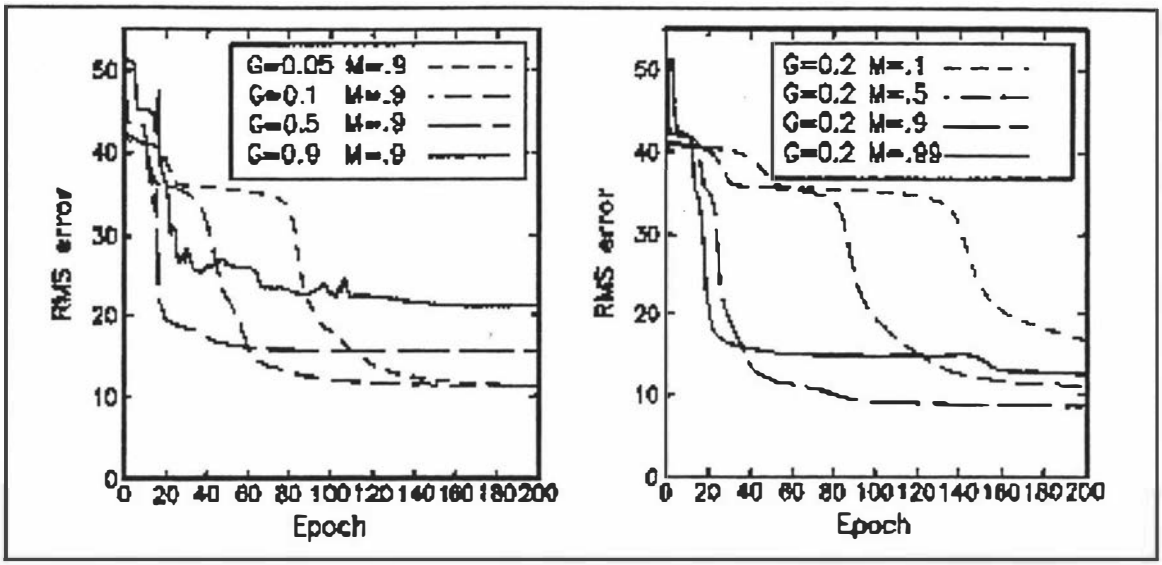


Figure 3.11: Effect of Gain and Momentum terms on convergence

The effect of varying the number of hidden units was also investigated. The results of training a network with 1, 3, 9, 25 and 100 hidden units is shown in Figure 3.12. It is interesting to note that up to some optimal number, which appears to be around nine in this case, both faster convergence and lower final error were produced by increasing the number of hidden units. Both the RMS errors and the peak errors are plotted. As the computation time for a single processor simulation of the network increases almost linearly with the number of hidden units, it was important to find a reasonably optimum number of neurons. The number of hidden neurons determines the complexity of the filtering operation which can be performed. Three or four hidden units was found to be adequate for emulating the first two filters. For emulating the Marr-Hildreth operator nine hidden units was found to be a more suitable choice.

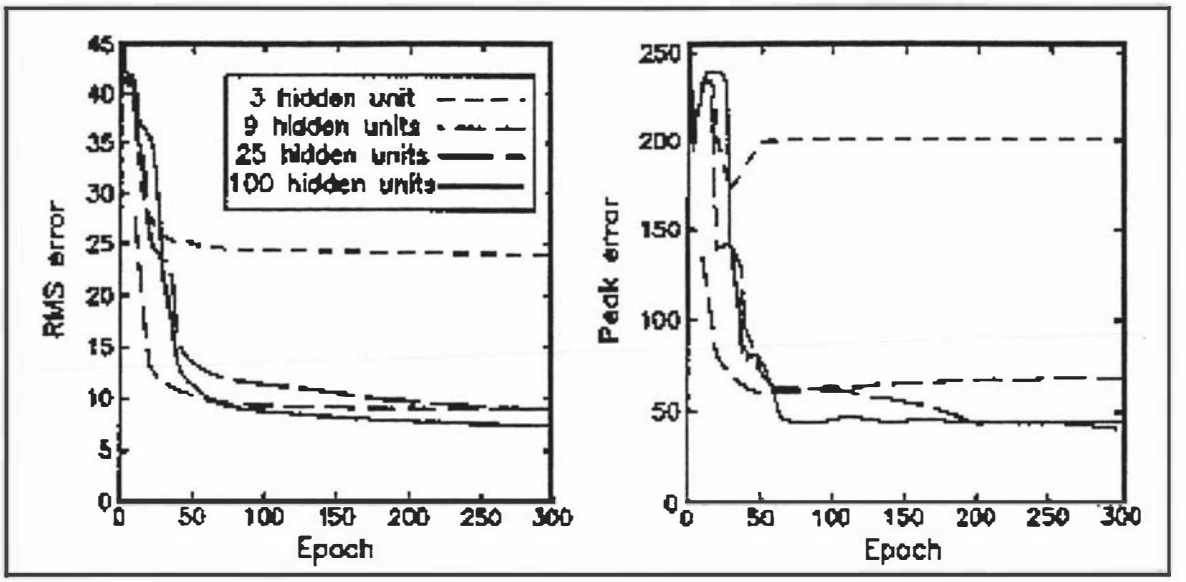


Figure 3.12: Effect of the number of hidden units on convergence

The algorithmic implementation of the Marr-Hildreth operator takes 63 seconds of CPU time to filter a 256x256 image on a MicroVax II. The trained NNWF takes 300 seconds (with 9 hidden units and a 7x7 input window), however the NNWF is coded for generality not speed and is inherently parallel in nature so it could more easily be moved to a parallel machine.

3.3 Further Experiments with the NNWF

Another common application for window filters is the removal of various types of noise. As an additional example of the use of the NNWF a series of experiments were performed in which the network filter was trained to remove a variety of types and combinations of noise.

3.3.1 Using the NNWF to remove noise.

One method of providing a target for the NNWF is to reverse the normal sequence of operations and generate the input from the target. For example, a noise free image may be used as a target and the input image generated by adding noise. The trained network filter can then be used to remove noise from other similarly degraded images.

Figure 3.13: Image degradation with different types of noise

1. Lena image.	**2.** Lena with addition of Gaussian noise with sigma of 32
3. Lena with addition of salt and pepper noise (10% pixels set to white and 10% set to black)	**4.** Lena with Gaussian additive noise with sigma of 15, 6% salt and pepper noise and signal dependent noise with standard deviation of $\sqrt{s}$

As an example of this method of using the NNWF the following series of experiments were performed. Two images were degraded with a variety of types of noise; Gaussian additive noise, salt and pepper noise, signal dependent noise and a combination of the three. The combination was of the following form;

$$s = s + \sqrt{s} \times \eta_1 + \eta_2 + sp \quad \text{.....3.5}$$

where η_1 and η_2 are mutually independent Gaussian white noise variables. η_1 provides signal dependent noise and has a variance of 1, η_2 provides additive noise with a standard deviation of 15(assuming a 0-255 grayscale image), sp represents salt and pepper noise with 6% of pixels set to white and 6% set to black. This mixture is suggested as a useful general noise model by Jain [Jain 1989 ibid. p273]. The resulting degraded images are shown in Figure 3.13. The two images used were the Lena image shown in Figure 3.13 and the Einstein image used in the previous experiments.

A section of one of the degraded images was then used as an input for training the NNWF with the corresponding section of the original image used as a target. After training the network in this way the network filter was applied to the degraded form of both full images. The trained network performed in a fairly similar way to standard noise removing filters such as the median filter. Figure 3.14 shows a comparison between the output of the trained network filter and the output of a median filter.

Type of noise	Degraded Image	NNWF Output	3x3 Median Filtered
Add Gaussian	31.7	12.4	14.6
Salt & Pepper	42.7	7.0	6.6
Mixed Noise	33.7	13.3(12.7)	12.7

Table 1: RMS Errors after noise removal

A network with a 3x3 window and a single hidden layer of nine neurons was used for these experiments. The training used a 64x64 sub-image. The network was trained for 1000 epochs for all but the salt and pepper noise. For the salt and pepper noise the network stabilized with an RMS error of 18 quite quickly, however after 3500 epochs of training it suddenly fell to a new plateau of 7. The removal of the black and white pixels then appears to succeed except where two or more faulty pixels occur within a 3x3 neighborhood. Table 3.1 shows the RMS differences between the original and degraded images and also

between the filtered and original images. The results show similar performance levels for the network and median filters. The second figure for mixed noise(12.7), shown in brackets, was obtained by pre-training on just salt and pepper noise. It is interesting to note that the training time required for the salt and pepper noise was the longest. As this is a situation in which a simple rule works well (leave the pixel alone unless it is 0 or 255) this may seem surprising. However, implementing this rule as a set of weights may be more difficult. The sudden discontinuities may make it more difficult to learn a suitable transform.

Figure 3.14: Noise removal with NNWF and median filter

1. Removal of additive noise using NNWF	2. Removal of additive noise using 3x3 median filter
3. Removal of salt and pepper noise by NNWF	4. Removal of salt and pepper noise by 3x3 median filter
5. Removal of combined additive, salt & pepper and signal dependent noise by NNWF	6. Removal of combined additive, salt & pepper and signal dependent noise by 3x3 median filter

3.4 Improving the performance of the Neural Network Window Filter

In later chapters of this thesis a technique for improving the performance of the neural network window filter is developed. This technique is based on the design and use of an educational environment. Before developing this approach it is appropriate to consider the following preliminary issues;

- What improvements to the NNWF are desirable for general image processing tasks?
- What other machine learning techniques are available and relevant to improving the performance of the NNWF?
- What improvements to back-propagation and what other neural network techniques are available and relevant?

Each of these questions is discussed briefly in the following sections.

3.4.1 Improvements to the NNWF for Image Processing[3]

What are the difficulties in applying the NNWF in the form in which it has been described and how can the range of situations in which it is useful be extended? These two questions are considered in the following paragraphs.

3.4.1.1 Limitations of the NNWF as described and implemented

In its present form, the training of the NNWF is time consuming taking several hours on a DEC Station 3100 for a typical training run of one thousand epochs. Time to satisfactory convergence is also variable taking from a few dozen to many thousands of epochs. The convergence sometimes proceeds in a series of

[3] Improvements to the NNWF can be considered from two perspectives depending on whether it is seen as an image processing tool or as a vehicle for studying a type of machine learning. This thesis covers both aspects, however it is from the first perspective that improvements are discussed here.

plateaus which may occur at widely spaced epoch intervals making it difficult to decide when to terminate training.

The configuration of a suitable network for a given task is at present an intuitive process relying on the experience of the investigator to select a suitable window size, a network structure and a suitable number of hidden nodes.

The neural network simulation and the image capture and display phases are currently implemented on different computers. Integration into a single package would simplify operation although it might not provide such a flexible development platform.

3.4.1.2 Extending the NNWFs usefulness

The following conditions can preclude the use of the NNWF for a particular task;

- Lack of a training image or a means of generating one.

  Additional methods of generating training images are suggested in the discussion at the end of this chapter and developed in later sections of the thesis.

- The required operation is not based solely on local information.

  Extensions to the NNWF to include non local inputs can allow the filter to be used in situations where some global information is required. If the information required to perform the task is not predominantly local in nature then it is probably better not to use the NNWF, although neural network techniques may still be applicable.

- It is not a filtering operation.

  Many image processing tasks are not filtering operations. For these tasks while neural network techniques may be applied the NNWF is not the appropriate tool.

- The NNWF takes too long to train, too long to run or does not converge to an acceptable minima.

Not converging to an acceptable minima in a reasonable time can either imply that the problem is insoluble or that a learning system with better performance is required. The two main avenues for improving performance in any machine learning system are;

- Designing a better learning device(a better pupil).
- Providing a better training environment(a better school).

The first of these is the subject of the next few paragraphs. The second is the approach which is developed in subsequent chapters of this thesis and represents a new contribution to the use of neural networks in this field.

A final limitation of the NNWF could be failure to generalise a particular filtering operation. If a specific data set is learnt rather than a general operation then the trained filter cannot be used to process similar images. This has not been a major problem in the experiments conducted so far, however one technique which can be used to combat this "over-training" is cross validation which is described in the following section.

3.4.2 Machine Learning and the NNWF

Many different approaches to machine learning and artificial intelligence are being actively investigated today. These approaches include; expert systems [IEEE Expert], artificial neural networks [IEEE Neural Networks, Neurocomputing, Neural Networks], fuzzy logic [Self 1990], genetic algorithms [Goldberg 1989], A-life [Langton et al. 1991] and numerous combinations of these. A considerable amount of work is now appearing in which attempts are made to usefully combine these various approaches to machine learning [Mitra & Pal 1995, Sestito et al. 1993, Kasabov & Jain 1993]. It is clear that each of these types of machine learning has encapsulated some aspect of the abilities and methods used in biological systems. Their integration appears to be a major task in the pursuit of artificial intelligence.

The combination of some of these other approaches with the neural network used in the NNWF would be one avenue for improving performance. This has not been investigated except as part of the general development of an educational environment. The use of rules in the training environment, which is

considered in later chapters, is one method of combining some of these techniques.

3.4.3 Other Neural Network Techniques and the NNWF

Within the field of artificial neural networks there are many different approaches and techniques. Multi-layer feed forward networks with back-propagation learning, while the most used, constitute only a small part of the field of artificial neural network research. In addition to other networks which are basically similar in form to the BP network but which use different learning schemes there are several other distinct categories; self organising networks [Kohonen 1984], associative memory networks [Beale & Jackson 1990], radial basis networks [Mak et al. 1994], hand crafted networks for modeling sections of natural neural systems [Rybak et al. 1992]. These are not discussed in detail here but some of these techniques could be used with the NNWF network to advantage. The use of a self-organizing network for pre-processing inputs is considered as part of the design of a training environment in chapter 7 [see §7.2]. Other networks, such as radial basis networks could also be used in a similar way.

Other networks such as the cascade correlation networks [Fahlman & Lebierer 1990, Hoehfeld & Fahlman 1992] could also form the basis of a NNWF. However, these are all methods of improving the pupil which while providing significant benefits should be considered in addition to improving the educational environment, not as a replacement for it.

Multi-layer feed forward networks with back-propagation learning have been one of the major areas for research in the field of artificial neural networks. Many variations and improvements to the technique have been reported over recent years. A few of these provide significant, generally applicable, modifications which are now used routinely by those working in the field. One of the difficulties in evaluating the many variations and additions described, however is the variety of tasks which have been used for testing these variations. The test tasks used range from completely contrived digital tasks such as the encoder problem[Lister 1992] through to real world analogue tasks such as controlling a chemical plant or recognizing handwriting. Techniques which provide considerable benefit in one task domain may not be useful or may actually be detrimental, when applied in other task domains. There is also a danger of analysing simpler, non realistic, situations when trying to find methods

of speeding up back-propagation [Wasserman 1993]. For this reason higher order methods which assume particular shapes for the error surface [Press et al. 1988] are often more likely to be fragile and perform unexpectedly when applied to real tasks. For these reasons it is appropriate to be somewhat conservative in applying modifications to BP while studying the use of an educational environment.

A number of the key modifications to standard back-propagation learning are described in the following paragraphs. Several of the modifications described here provide alternative methods of dealing with a basic problem alluded to in the description of steepest descent learning given in chapter two [§2.2.2]. Using a step size which is proportional to the gradient, while conservative and stable, produces small steps when traversing large almost flat plains in the error surface. This leads to very slow convergence in such situations.

3.4.3.1 Alternative error functions

The error function which is normally used in BP networks is the sum of squared errors. The summation is taken over all outputs for all output-target pairs in a training set. This function has been chosen somewhat arbitrarily. For a particular task other error functions may be more appropriate. For example, errors at one particular output unit may be of greater importance than errors at the other output units. If this is the case then the error function should reflect this. The error function chosen also defines the relative importance of errors of different magnitudes. It is normally desirable that the error measure should be independent of the sign of the target-output difference, using squared errors is but one way of achieving this. A squared error also penalises large errors more than small errors (one error of 4 is equivalent to 16 errors of 1). This may not be appropriate particularly when binary targets are being used.

If outputs are real values with added Gaussian noise and all outputs are of equal importance then the squared error is the correct function [Hinton 1993]

Van Ooyen has suggested an alternative error function which avoids the problem of slow convergence which occurs when any neuron output is near to saturation in the wrong direction [Ooyen & Nienhuis 1992]. The alternative error function is given by

$$E = \sum_p E_p \qquad E_p = \sum_k \left[t_{kp} \ln o_k + (1 - t_k) \ln(1 - o_k) \right] \quad \text{.....3.6}$$

and results in equation 2.6 simplifying to

$$\delta_k = (t_k - o_k) \quad \textbf{3.7}$$

This alternative error function effectively removes the o(1-o) term in the weight update equation for the final layer of a BP network.

While this appears to be a promising approach it could not easily be incorporated into the structure of the AM package. For this reason and as it was not essential to the development of the main theme of this thesis it was not used in the NNWF experiments.

3.4.3.2 Alternative squashing functions

The back-propagation algorithm seems to be surprisingly unaffected by the form of the squashing function provided it is non linear, differentiable and bounded. Networks with quite different squashing functions still reach a similar level of performance even though the required set of weights will, in general be quite different. Some of the experiments reported above were repeated using a network with a sin(x) squashing function. This had very little effect on the results. Hinton suggests that symmetrical squashing functions, such as the hyperbolic tangent $\tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$, will usually lead to faster convergence [Hinton 1993 ibid. p3.26].

3.4.3.3 Alternative weight initializations

In the basic implementation of a BP network all weights are initialized to small random numbers. It is necessary that the initial weights are different to allow the different neurons to form different functions during training. In addition small weights are required to avoid saturated neurons which can lead to slow convergence as described above. Small initial weights can also be considered as a method of limiting the network complexity during the initial stages of learning as neurons with small weights will be operating in the linear portion of the squashing function. As any multi-layer network with linear neurons can be replaced by a single layer network, BP networks with small weights are equivalent to a single layer network.

For the specific case of a two layer network in which the second layer contains linear nodes an alternative initialization technique has been found by Nguyen

and Widrow [Nguyen & Widrow 1990]. This is not applicable to the networks used in the NNWF experiments which do not use linear nodes in the final layer and in many cases use three layer rather than two layer networks. A NNWF could, however, be built using this style of network and weight initialization.

3.4.3.4 Conjugate gradient

The steepest descent technique used in back-propagation only uses first derivative information to determine weight changes. If something is known about the general shape of the error surface then it should be possible to use information from higher derivatives to increase the rate of convergence. The conjugate gradient method is one such approach. It assumes a quadratic error surface and proceeds in a series of line minimisations along approximately conjugate directions. This assumption while generally approximately correct near a minima is unlikely to be valid elsewhere. The line minimisation for a BP net consists of finding the best gain term to use for a given weight change direction. [Møller 1993] A direction **v** is conjugate to a direction **u** if any changes in gradient along the line **v** are perpendicular to **u**.

Conjugate Gradient is normally used with epoch weight update rather than pattern weight update. Conjugate gradient methods are available as an option in the AM package used to implement the NNWF, however, they require all training patterns to be stored and available in different areas of memory at the same time. This is not normally possible with the NNWF as it would require storage for $w^2(x-w+1)(y-w+1)$ elements at run time. This is because the w^2 inputs for each window position are largely a repeat of the inputs for the previous window position. These are not normally stored separately but extracted from the image as required. Conjugate gradient methods were investigated but have not been used in the window filter experiments for this reason.

3.4.3.5 Quickprop

Quickprop which was developed by Fahlman was one of the early attempts to speed up back-propagation [Fahlman 1988]. It is an approach which tries to estimate and use second derivative information to speed up convergence. It also incorporates a number of heuristics to limit the increase of step size, modify the error measure for large errors and avoid saturated neurons stopping weight changes. It was tested by Fahlman on a variety of N-M-N encoder problems on

which it provided considerable speedup in comparison with standard BP. The encoder problem, however, is a rather unusual class of problem in which similar inputs should produce very different outputs. Neural networks are often required to generalise and hence produce similar outputs for similar inputs. Due to this important difference and the addition of further parameters which must be chosen appropriately it is hard to compare the performance of Quickprop in any general way. Its main importance may have been to stimulate research into methods of speeding up back-propagation learning.

Another technique found helpful by Fahlman is the use of different values of gain dependent on the fan-in of the units. This is often used as part of other techniques. This is not easily implemented in AM and was therefore not used in the NNWF experiments.

3.4.3.6 RPROP

This technique developed by Riedmiller removes the dependence of step size on gradient magnitude [Riedmiller & Braun 1993]. The partial derivatives are only used to determine whether to change the weight and if so in which direction. The size of the change is handled by a heuristic which determines step size by comparing the last two partial derivatives. If both are of the same sign then the step size is increased slightly, if they are of opposite sign then the step size is decreased drastically. This is done on a weight by weight basis. This has not been used in the NNWF as it would be difficult to incorporate it into the AM package. However similar advantages can be produced by using dynamic weight updates which are discussed below.

3.4.3.7 SuperSAB

SuperSAB is another technique which modifies step sizes in an attempt to improve the convergence rate [Tollenaere 1990 ibid.]. It is based on the following set of ideas;

1. Step sizes should be adjusted during training on a weight by weight basis.

2. If the weight change at one step is in the same direction as at the previous step then the step size for that weight should be increased.

3. If the weight change for one step is in the opposite direction to that taken in the previous step then the last step should be undone and the step size should be decreased as the minimum has been skipped over.

The exact size of the increase and decrease parameters is not critical to the operation of this algorithm. SuperSAB is often 10-100 times faster than standard BP although again this is problem dependent. Again this is a technique which could not easily be added to the AM package so it was not used for the NNWF experiments.

3.4.3.8 Dynamic weight updates

As it is computationally easier to use the gradient as the step size, a simpler approach to controlling step size is to dynamically vary the gain term, G in equation 2.4. This approach modifies the gain after each set of weight updates using the following procedure. If overall error has increased then the gain should be dramatically decreased (typical a factor of 0.5 is used), some implementations also throw away the last set of weight changes if the error has increased by more than a critical ratio (e.g. 1.04 in MATLAB NN Toolbox [Demuth & Beale 1992 ibid.]), however, this requires a second set of weights and biases to be stored. If the overall error is less than the previous one then the gain is increased slightly (a factor of 1.1 is typical).

A very similar approach to this had already been added to the NNWF which automatically adjusted the gain term after each training epoch. The gain was halved if the error had increased. If error had decreased then the gain was increased by ten percent, A limit was also set on the minimum gain to be used, typically 0.005.

3.4.3.9 Stopping Learning Early

Normally a training set represents a small sample from a large input space with corresponding target outputs, perhaps contaminated with noise. It is hoped that during training the network will learn to generalize the required input/output transformation. Training to a perfect performance level on a small sample may actually decrease performance on the full set. The cross validation technique consists of dividing the training set in two and using one half for training and the other for measuring expected performance on the full data space [Hansen & Salmon 1990 ibid.]. Training is stopped when the performance on the validation

set starts to decrease. This prevents "over training" which can lead to poor generalization.

The NNWF training system was modified to allow either a second part of the same image or a part of a second image to be used for cross validation during training.

3.4.3.10 Discussion

Many of the techniques described above tackle a basic flaw in the steepest descent method; the flaw is *making the step size dependent on the magnitude of the error gradient.* This fault can be counteracted by modifying a gain factor (by which the gradient vector is multiplied) or by determining the weight change magnitude itself by some other heuristic as is done in RPROP. Which of these approaches is taken is probably of little importance. As the gradient must be determined to find its sign, no computational overhead is involved in using it. Keeping it as a factor also ensures that the system will be stable. A second factor for consideration in all these methods is whether a single global gain is to be used or whether each weight should have its own step size. The first approach is less computationally intensive but may not provide the full benefits to be gained by removing the dependence of step size on gradient. The second approach involves more storage and also removes the steepest descent aspect of BP. If the weights are not changed in proportion to the components of the gradient vector, the direction while down hill may not be in the direction of steepest descent. In this way these techniques may be considered similar to conjugate gradient methods which attempt to determine a more efficient set of movement directions.

A second "flaw" in the standard steepest descent method is the universal assumption that a quadratic error function is appropriate. At present, however, there appears to be no general way of determining the appropriate error function for a given task [Elgerd 1967].

Conjugate gradient and Quickprop make assumptions about the shape of the error surface. Near minima these assumptions are generally valid, or reasonably so, elsewhere they are unlikely to be correct, and hence the techniques are often only used once initial convergence has occurred.

This thesis is concerned with improving how well or how quickly a task is learnt by suitably adjusting the training environment. The main concern therefore is

what constitutes **suitable manipulation** of the training environment. Therefore the choice of a learning algorithm is relatively unimportant, provided the same technique is used in each experiment. In the remainder of this thesis the main concern has been whether a particular transform can be learnt rather than how fast it can be learnt. Some of the assumptions inherent in the speedup techniques discussed above may make them more fragile. While this is partly speculation at this stage, in general terms pursuit of a single goal (in this case convergence speed) often produces compromises in other areas. For this reason for the early stages of the investigation of the use of a training environment it is not appropriate to use speedup techniques. To do so could lead to confusion as the educational environment may produce improvements by counteracting deficiencies introduced by the speedup techniques.

3.5 DISCUSSION

1. In the remainder of this thesis methods relating to the improvement of the "educational environment" are developed and described. The methods of improving the pupil have been briefly sumarised above and should, in general, be considered as an adjunct rather than an alternative to the work which is described in the following chapters.

2. The experiments described in this chapter represented a preliminary study of a neural network based universal window filter. They show that the NNWF can learn filtering operations from representative input-target image pairs and, at least in the situations described, generalise the required operation. These results suggested further study of this technique was warranted. Very few problems with local minima were encountered. In general if a network did not converge to an acceptable minima restarting with a new set of random weights did not help.

3. The small size of the networks which were required to perform these transformations suggest that a hardware implementation of the NNWF could be practical. A relatively small number of inputs and hidden nodes are used in these networks. The most complex network used routinely in the work described in this thesis had only a 9x9 input window and two hidden layers containing five and three neurons respectively. This had the consequence that simulations could be performed in a reasonable time on the computer systems available.

4. If the filter network described is to be considered as a universal window filter the creation of targets needs further consideration. How can target images be provided in the more general case where an existing algorithmic technique is not being emulated? A number of techniques can be considered;

   Target creation by a domain expert.

   This could be done entirely by hand or by using a variety of image processing and computer graphics techniques to construct the target. The capability of the NNWF to learn subjective distinctions should be an advantage in this situation as it takes advantage of the non-algorithmic nature of neural networks. A final system can be envisioned in which a

sample image or part image would have the features of interest marked on it by a domain expert. These image pairs would be used to train the NNWF filter. The remaining images would then be classified automatically.

Target creation by an image processing expert

An image processing expert could be used to select and apply a sequence of operations to a sample image to provide a suitable target image for training. This may either be constrained to only include non-subjective steps or may include subjective evaluations (for example choosing a threshold level which best suits this particular image.

Target creation by special one-off procedures

Using a slower or more complex image capture or image creation procedure to produce a required target. For example if the NNWF is to remove noise due to the short integration time which is required in a particular situation then it may be possible to provide extra lighting or multiple exposures for the construction of the target. Naturally if the image can be improved easily and routinely in this way then this would be preferable to later filtering with the NNWF. It is only in the case where the extra effort or time can be justified only during the training that this technique is applicable.

Target creation by reversing the normal flow.

In some situations it is possible to reverse the normal flow of operations and produce the input image from the target image. For example, starting with a sharp image which will be used as a target a blurred, or noisy input image can be created. The NNWF could then be trained using this pair to remove blur or noise. The experiments in section 3.3 above provide an example of this method of training the NNWF.

5. Even though the emulations of the window filters were successful they were not perfect and in the case of the Marr-Hildreth considerable improvement could be sought. The two main approaches to improving the performance are; changing the network and changing the training. The first of these has

already been discussed in section 3.4 the second is developed in chapter five and later chapters of this thesis.

6. Other possible extensions to the basic structure of the NNWF include;

    - input windows which are non-square
    - multi-resolution input windows
    - augmenting the inputs with other types of information such as image statistics or window position.
    - Using parts of the output image already scanned to augment the network input.

    While this work is not drawn on in the rest of the thesis some preliminary experiments on the last three extensions mentioned above has been done. This is briefly described here;

    - The NNWF was modified to incorporate a multi-resolution window in which a central 3x3 pixel window was surrounded with a ring of eight regions each region being 3x3 pixels, This was in turn surrounded by a ring of regions each 9x9 pixels in size. The intention of this arrangement was to provide fine detail information in the centre and then gradually more general or averaged information as the distance from the centre of the window increases. A NNWF of this form was implemented and was able to learn filtering operations. This form of the NNWF has not been applied to any significant tasks as yet.

    - In order to enable the NNWF to perform operations in which the position within the image should influence the output the NNWF network inputs were augmented with the x and y coordinates of the centre of the window. This was used to train the NNWF to remove image distortion from a set of similarly distorted images.

    - To allow the NNWF to learn to favour the production of continuous lines in the output images the NNWF was modified to augment the network inputs with the pixel values in a corresponding window in the output image. The window used did not include values from below or to the right of the central pixel. Only initial experiments have been performed with this NNWF implementation.

The investigation of non square windows and further exploration of the variations discussed above has been left as possible future work. It was decided at this point in these investigations to concentrate research on improving the educational environment. Other areas worthy of further investigation include; finding methods of improving the network(or pupil), extending the basic structure of the NNWF and producing parallel or hardware implementations of the network filter. These all involve interesting possibilities but were not pursued as part of this thesis.

3.6 REFERENCES

page first cited

Bailey & Hodgson 1988
D.G. Bailey, R.M. Hodgson, *VIPS - A digital image processing algorithm development environment*, Image and Vision Computing, Vol. 6 No. 3, August 1988, pp176-184 ..... 3.4

Beale & Jackson 1990
R. Beale, T. Jackson, Neural Computing an Introduction, Adam Hilgar, Bristol, 1990, Chapter 8 ..... 3.29

Elgerd 1967
O.I. Elgerd, Control Systems Theory, McGraw Hill, New York, 1967, pp296-301 ..... 3.35

Fahlman & Lebierer 1990
S.E. Fahlman, C. Lebierer, *The cascade correlation learning architecture*, in Advances in Neural Information Processing Systems 2, Morgan Kaufmann, Los Altos CA, 1990, pp524-532..... 3.29

Fahlman 1988
S.E. Fahlman, *Faster-Learning variations on backpropagation: An empirical study*., in Proceedings of the Connectionist Models Summer School, Morgan Kaufmann, San Mateo CA, 1988, pp38-51 3.32

Goldberg 1989
D.E. Goldberg, Genetic Algorithms in Search, Optimization & Machine Learning, Addison-Wesley, New York, 1989..... 3.28

Hansen & Salmon 1990
L.K. Hansen, P. Salmon, *Neural Network Ensembles*, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 12, No. 10, October 1990, pp993-1001..... 3.7

Hinton 1993
Geoff Hinton, Tutorial on Neural Networks, University of Sydney, Sydney, Feb 1993, p2.20... 3.30

Hoehfeld & Fahlman 1992
M. Hoehfeld, S.E. Fahlman, *Learning with Limited Numerical Precision Using the Cascade-Correlation Algorithm*, IEEE Transactions on Neural Networks, Vol. 4, 1992 ..... 3.29

IEEE Expert
IEEE Expert: intelligent systems and their applications, IEEE Computer Society, Los Alamitos Calif., 1995..... 3.28

IEEE Neural Networks
IEEE Transactions on Neural Networks, IEEE Neural Networks Council, New York, 1995 ..... 3.28

Jain 1989
A.K. Jain, Fundamentals of Digital Image Processing, Prentice & Hall, Englewood Cliffs NJ, 1989, pp355-356..... 3.15

Kasabov & Jain 1993
N.K. Kasabov, L.C. Jain, *Connectionist Expert Systems*, in Proceedings of Artificial Neural Networks and Expert Systems, Dunedin, 1993, pp220-221..... 3.28

Kohonen 1984
T. Kohonen, Self-Organization and Associative Memory, Third Edition, Springer Verlag, Heidelberg, 1984 ..... 3.29

Langton et al. 1991

C.G. Langton, C.E. Taylor, J.D. Farmer, S.R. Rasmussen, Artificial Life II, Addison Wesley, Redwood Calif., Feb 1991 ..... 3.28

Leighton 1991

R.R. Leighton, The Aspirin/MIGRAINES Software Tools: Users Manual, The MITRE Corporation, 1991 ..... 3.5

Lister 1992

R. Lister, *Back Propagation and the N-2-N Encoder Problem*, Proceedings of the Third Australian Conference on Neural Networks ACNN'92, Sydney, Feb 1992, pp198-201 ..... 3.29

Mak et al. 1994

M.W. Mak, W.G. Allen, G.G. Sexton, *Speaker identification using multilayer perceptrons and radial basis function networks*, Neurocomputing, Vol. 6 No. 1, 1994, pp99-117 ..... 3.29

Mitra & Pal 1995

S. Mitra, S.K. Pal, *Fuzzy Multi-Layer Perceptron, Inferencing and Rule Generation*, IEEE Transactions on Neural Networks, Vol. 6 No. 1, Jan 1995, pp51-63 ..... 3.28

Møller 1993

M.F. Møller, *A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning*, Neural Networks, Vol. 6 No. 4, 1993, pp525-533 ..... 3.32

Neural Networks

Journal of the International Neural Network Society, Pergamon Press, 1995 ..... 3.28

Neurocomputing

Neurocomputing Journal, Elsevier, Amsterdam, 1995 ..... 3.28

Nguyen & Widrow 1990

D. Nguyen, B. Widrow, *Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights*, International Joint Conference of Neural Networks, July 1990, Vol. 3 pp21-26 ..... 3.32

Ooyen & Nienhuis 1992

A. Van Ooyen, B. Nienhuis, *Improving the Convergence of the Back-Propagation Algorithm*, Neural Networks, Vol. 5, No. 3, 1992, pp459-463 ..... 3.30

Press et al. 1988

W.H. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling, Numerical Recipes in C 3.30

Riedmiller & Braun 1993

M. Riedmiller, H. Braun, RPROP: *A Fast and Robust Backpropagation Learning Strategy*, in Proceedings of the Fourth Australian Conference on Neural Networks ACNN'93, Melbourne, Jan 1993, pp169-172 ..... 3.33

Rybak et al. 1992

I.A. Rybak, N.A. Shevtsova, V.M. Sandler, *The model of a neural network visual preprocessor*, Neurocomputing, Vol. 4, 1992, pp93-102 ..... 3.29

Self 1990

K. Self, *Designing with fuzzy logic*, IEEE Spectrum, 1990, P42-45&105 ..... 3.28

Sestito et al. 1993

S. Sestito, S. Goss, G. Merrington, R. Eustace, *Automated Acquisition of Rules for Diagnosis*, in Proceedings of Fourth Australian Conference on Neural Networks ACNN'93, Melbourne, 1993, pp150-153 ..... 3.28

Tollenaere 1990

T. Tollenaere, SuperSAB: Fast adaptive back propagation with good scaling properties., Neural-Networks, Vol 3, 1990, pp561-573 ..... 3.18

Wasserman 1993

P.D. Wasserman, Advanced methods in neural computing, Van Nostrand Reinhold, New York, 1993 ..... 3.30

Chapter 4[1]

4. Neural Networks and Intelligent Systems

This chapter contains the most speculative aspects of this thesis. In an attempt to identify key features of intelligent systems, hypothetical designs are considered for creatures which are to exist in a slowly changing, and possibly hostile, environment. The term logical sequential thought is defined and a possibly original suggestion is made for its function in mankind. The frame problem and expert systems are also discussed from this perspective.

The ideas presented in this chapter provide many possible avenues for future experimental research. One of these relates to the suggested importance of logical sequential thought for education rather than for problem solving. This idea led to the proposal for using an educational environment to improve the learning performance of neural networks. The development and implementation of such an educational environment is described in the remainder of the thesis.

---

[1] This chapter is intended to be read as a speculative essay, it is relevant but not pivotal to the thesis. It is incorporated to give an account of the speculations that led to the development of the education/training environment for the artificial neural network based window filter. It encompasses some areas of scholarship outside the area of specialist expertise of the author. However, the arguments presented have the merit of being at least plausible and have, at the very least, generated useful insights.

4.1 INTRODUCTION

The previous sections of this thesis contain a discussion of the design and implementation of a neural network based window filter. A series of experiments are then reported in which the network filter is trained to perform useful tasks. In moving to more complex tasks it was found that the network would sometimes not converge to an acceptable solution or would take a very long time to converge. This naturally led to consideration of the possible causes and cures for the poor performance. In the final section of the previous chapter, conventional methods of improving the learning performance of neural networks are described. These methods focus on improving the network rather than the system as a whole. Other researchers have considered combining NNs with other techniques such as the use of genetic algorithms and expert systems. This appears to have been done on an empirical basis with little philosophical consideration as to why such combinations might be beneficial.

In the search for methods of improving the training and general usefulness of neural networks it is appropriate to consider the philosophical and psychological basis of intelligence and intelligent systems. A rather general and speculative approach to intelligent systems and their development is introduced in this chapter. The first section of this chapter contains background material in which the term logical sequential thought (LST) is introduced and its importance in intelligent systems is indicated. In the next section an alternative approach to the design of intelligent systems is introduced. The use of logical sequential thought is considered in a framework of a series of increasingly sophisticated designs for artificial creatures. This leads to an alternative interpretation of the uses of LST and rules in intelligent systems. The intriguing frame problem is then described and analysed with reference to these ideas. The success and failure of expert systems are also discussed in this context.

These speculations on intelligent systems led to the idea of developing an educational environment for improving the performance and utility of neural networks. The development and evaluation of such an educational environment for the NNWF is then described in the remaining chapters of the thesis.

4.2 BACKGROUND

Life has evolved from simple single cell creatures barely able to respond to their environment through to humankind, a very complex animal that not only responds to the environment but also modifies and controls it.

At various times people have tried to duplicate and improve upon the abilities of man and various other animals. These duplications or improvements have become known as machines. First the physical abilities were reproduced leading to the industrial revolution and more recently, since the late 1940s, various attempts have been made to reproduce in electronic form the mental abilities of man. One interesting feature of these attempts is the mixture of success and failure. In some areas considerable success has been achieved; a modern calculator can handle arithmetic far more quickly and accurately than a man. In other areas such as pattern recognition, speech recognition or general problem solving the results have been almost pitiful. It is very difficult to build an artificial system which can emulate the abilities of simple animals let alone mankind.

In trying to build an intelligent machine, it was perhaps natural for researchers to take the pinnacle of mankind's evolved abilities as the goal. The implicit principle being that, if a machine can be produced which can do the things that people find most difficult, surely it would be intelligent, and surely it would be able to do all the other things people find easy. Many examples of this general approach are available [see Collins & Smith 1988 chapters 1&5, Newell & Simon 1972]. The pinnacle of mankind's abilities appears to be logical sequential thought (LST). The line of argument is essentially as follows; "Our ability to use a sequence of logical steps to solve problems is surely the very essence of intelligence, therefore it is this which must be duplicated if intelligent machines are to be produced."

More formally the term logical sequential thought, or rule based deductive reasoning, is used here to refer to the process of applying a logical sequence of steps to deduce a problem solution or best course of action for a given situation. The deductions are made from known facts, pieces of sensory information and knowledge in the form of explicit rules. Scientific deduction, as it is supposed to occur, is perhaps the prime example of this process. Mathematical deductions or proofs of theorems are a more stylized and abstract version of LST.

4.3 An Alternative Approach to Intelligence

Recent work on artificial neural networks (ANNs) has led to an understanding of how simple conditioning [Card & Moore 1990] and associative memory [Kohonen 1984] can be reproduced in electronic form, however there still appears to be a considerable gap between these sorts of abilities and what people do when they think and learn.

Back Propagation networks can be trained to produce a required set of responses to a variety of inputs [Rumelhart & McCleland 1986] but this requires a training set of *input-target* pairs. It is difficult to see how this can be achieved from random interactions with an environment. Reinforcement learning removes the requirement for immediate feedback [Sutton 1992] but still leaves a large gap between the properties of artificial neural networks and what would generally be described as the properties of intelligent systems. In trying to understand and reproduce intelligence rather than trying to duplicate the sequential thought process as such, it is useful to consider an alternative approach similar to that suggested by ALife studies [Langton et al. 1991].

Let us consider the design requirements for a creature which is to survive in a possibly hostile and slowly changing environment. The environment is to contain several populations of different "creatures." What attributes could the creatures have that would make them more likely to survive and reproduce, and hence continue to exist and flourish as a species?

The above considerations led me to suggest the following four levels, or types of creature to be analysed. These levels have been chosen to make some specific points about intelligent systems. They are not intended, and clearly do not, encompass all the rich behavioural complexity of real biological systems and their complex evolution [Manning 1979].

4.3.1 Creatures of a fixed design.

Creatures in the first level are "built" using fixed designs. Members of a particular species of this type may, for example, all have four legs and move towards anything that is green.. The key point about this type of creature is that it does not change from generation to generation. A particular species of creature at this level will be suited to a particular

environment and will continue to survive provided things do not change too much and provided the original "design" was reasonably well matched to the prevailing environmental conditions. If conditions change either gradually or quickly, level one creatures are likely to be at a disadvantage.

4.3.2 Creatures with genetic inheritance.

This second level of creature is one that has some variety within a species. It also has a mechanism for passing on information from generation to generation indicating which individuals, or parts of individuals, are well suited to the present environment. For example, if a shadow falling on the creature is often followed by being eaten, then those few members of the species who jump when a shadow falls on them will live longer and pass that trait on to their offspring. This is an improvement on level one creatures and in the long run level two creatures will replace level one creatures. This process of design change by natural selection is a slow one and it will take many generations to make the jump-at-shadows design, the norm.

4.3.3 Creatures that respond to conditioning.

The third level creature is one that can be conditioned. If eating blue-green things is repeatedly followed by feeling unwell then level three creatures will stop eating blue-green things. This can be combined with the level two approach and results in a creature more fit to survive. The conditioning depends on repeated pairings of events. Such conditioning may take place in simple creatures, such as the eater of blue-green things and in more complex creatures involved in higher level activities. For example, if the task is to spell words containing the letters i and e next to each other correctly then it might work as follows: Each time the creature needs to spell such a word it guesses, initially at random. If it is right then it is rewarded by silence. If the creature is wrong the spell checker beeps and it has to change the spelling. After a long time the creature's neural network is slowly trained to spell correctly. This form of design modification can take place more quickly than the genetic design change of the level two creature. However, the difficulty is every individual member of the species has to go through this same process, to use the

example above, they all have to experience feeling unwell after eating blue-green things to modify their design appropriately.

4.3.4 Creatures with logical sequential thought.

The fourth level creature can apply logical sequential thought. It does not have to use the environment directly. It does not need to waste time and effort consuming those nasty blue green things and getting sick or to endure the incessant beeping of the spell checker. It can use the experience of others who have told it that "i before e except after c provided the sound is e." The rule supplies the correct/required output and it can be used to train a neural network in a very similar way to the conditioning provided by the environment for level three creatures. An important point to be made is that logical sequential thought is not being used to solve problems or determine actions. It is being used to enable the level four creature to be conditioned without recourse to the real physical environment. Eating of blue-green things to slowly change behaviour is a dangerous process, eating too many of them may result in death which certainly reduces the benefits of learning. Just feeling unwell may reduce the creatures ability to deal with other important aspects of life. Using LST in times of relative calm to slowly condition a small network to jump at shadows is a relatively safe process. Later, when a shadow of the tiger falls across the optical sensor of the level four creature, it jumps. The immediate action is produced by the conditioned network. For the more complex creature that is learning to spell, the *i before e* rule has not been "learned" until the rule is no longer used. The required action is produced by a section of the creature which has now been conditioned.

These rules (eating blue-green things makes you feel unwell and i before e ...) which form part of the LST process have to be transferred from one creature to another. This implies some form of communication and will probably be easier if some form of language exists.

Evolutionary advancement is constrained to proceed in a series of steps each of which bestows some small survival advantage. This is a quite different process from the top-down design used in engineering projects. Both approaches have advantages and disadvantages, however the difference between the two can have important implications when attempting to transfer abilities from the biological world to an "artificial" world. Each feature which is to be transferred

needs to be considered in context. A feature which is beneficial in one context may not be so useful in isolation.

The discussion above concerning a progression of increasingly sophisticated creatures demonstrates one possible role for logical sequential thought. If it is used as an adjunct to the structures present in level two and three creatures, it will provide quite different results from when it is used alone. However, a feature which evolved for one purpose may well be used later for another. The creature which used LST to condition a sub-network might also use the LST directly, for example, instead of immediately starting to eat when faced with a possible food source it could; hesitate, determine present inputs, look for known rules containing references to blue green things, infer that eating will produce a feeling of being unwell, note that this does not fit with the aim of reducing the feeling of being unwell due to hunger, and **finally** move on in search of other food. There is a difficulty here in that using LST in this way involves a process which can be described as hesitation and, at least in the case of the shadow of a tiger, "He who hesitates is lost." [proverb anon.] A more formal description of this difficulty is to say that LST used in isolation leads to the frame problem. This is described in the next section.

4.4 THE FRAME PROBLEM

The frame problem of artificial intelligence is discussed by Daniel C. Dennett in "Minds, Machines and Evolution: Philosophical Studies [Dennett 1984]" by reference to a robot named R_1 whose only task was to fend for itself. The frame problem is one of deciding what items to consider when trying to predict the outcome of some action. Given finite resources there is not time to consider everything. The difficulty is in deciding which consequences need to be evaluated. This appears to be a problem which must be dealt with by any creature which is to be considered intelligent and is to prosper in a possibly hostile and slowly changing environment.

The example of the frame problem used by Dennett is as follows:

> *Once upon a time there was a robot, named R_1 by its creators. Its only task was to fend for itself. One day its designers arranged for it to learn that its spare battery, its precious energy supply, was locked in a room with a time bomb set to go off soon. R_1 located the room, and the key to the door, and formulated a plan to rescue its battery. There was a wagon in the room, and the battery was on the wagon, and R_1 hypothesised that a certain action, which it called PULLOUT(WAGON,ROOM) would result in the battery being removed from the room. Straightway it acted, and did succeed in getting the battery out of the room before the bomb went off. Unfortunately, however, the bomb was also on the wagon. R_1 knew the bomb was on the wagon and could have deduced that moving the trolley would move the bomb. It had not been programmed to consider the side effects of its actions.*
>
> *Back to the drawing board. 'The solution is obvious,' said the designers. 'Our next robot must be made to recognise not just the intended implications of its actions, but also the implications about their side effects.' The new robot R_2D_2 (robot deducer) was placed in the same predicament. It too hit upon the idea of PULLOUT(WAGON,ROOM), it began, as designed, to consider the implications of this act. It had just finished deducing that pulling the wagon out of the room would not change the colour of the room's walls, and was embarking on a proof of the further implication that*

pulling the wagon would cause its wheels to turn more revolutions than there were wheels on the wagon..... when the bomb exploded.

The frame problem is one of deciding what is relevant in a particular situation and has proved very difficult to overcome [Dreyfus 1993].

If we look at the assumptions underlying the description of the frame problem they seem to include:

a) A system that could use logical sequential thought should be considered intelligent. By "use logical sequential thought" we mean that it has knowledge of its environment and possible actions, a set of rules giving results of certain actions and a method of combining these to find the implications of proposed actions.

b) An intelligent being (the robot) should be able to look after itself, in this case avoid being destroyed by the bomb.

These two assumptions lead to the frame problem. What grounds do we have for either of these assumptions? If we accept them then the R_1 example clearly shows an intelligent being which cannot look after itself, but intelligent beings should be able to look after themselves, we are intelligent and we can look after ourselves, hence the paradox. Both of the assumptions are suspect but the key point is that we have no reason to assume that a faculty for logical sequential thought **by itself** is a survival trait. As an additional feature, used with the other abilities, it certainly has benefits but by itself it may be of no value. Even in conjunction with the level two abilities described previously, too much LST may be detrimental to survival. It could even be argued that too much dependence on LST must be detrimental to survival otherwise people would be better at it!

The paradox of the frame problem, like many other paradoxes, may merely be the result of faulty assumptions. It is possible to avoid problems with the first assumption (LST = intelligence) by regarding it as a definition of intelligence. I suggest this is a poor definition of intelligence, it merely devalues the word to match a method which it is hoped might produce intelligent behaviour. Even if we accept this definition the paradox of the frame problem still disappears. The question just becomes why should intelligence be sufficient to provide survival. Both assumptions are required to form the paradox.

Intelligence is a difficult word to define with definitions ranging from "That which is measured by intelligence tests " to more vague but in some ways more

satisfying definitions such as "General problem solving ability" or "Ability to survive in a complex world" [see Sternberg & Detterman 1986 or Boring 1923]. It is perhaps the difficulty of defining intelligence that has led to the formulation of the frame problem. When we find a word difficult to define it is easy to fall into the trap of choosing a simple definition (in terms of its formulation) but then using the word as though it still had the more generally accepted though ill defined meaning.

The human ability to survive in a complex changing world, where decisions must of necessity be made quickly and without full information may well depend on the "older[2]" parts of the brain which do not use LST. Conditioned neural networks which have been trained by repeated exposure to the real world do not need to decide which items are relevant, all past events are used implicitly in deciding actions. All past events are included in the network to the extent that the connection strengths between neurons have been shaped by experience.

It is true that we occasionally use a small amount of LST to decide on an action but we may well do this at the risk of falling into the frame problem. It has been noticed at least since Shakespeare's time (if not before) that those members of mankind who are too good at LST may not be good survival material.

> *And thus the native hue of resolution*
> *Is sicklied o'er with the pale cast of thought,*
> *And enterprises of great pith and moment*
> *with this regard their currents turn awry*
> *And lose the name of action*
> *(Hamlet,III,i,56)*

The existence and success of expert systems appears, at least initially, to be inconsistent with this theory of the purpose of LST. This issue is considered in the next section (expert systems).

---

[2] The term older is used in an evolutionary sense to refer to parts of the brain which evolved in the more distant past. A simplified description of this form is given by Ornstein & Thompson [Ornstein & Thompson 1986].

4.5 EXPERT SYSTEMS

Many different approaches have been tried in the attempt to create machines which duplicate the mental abilities of man. The expert systems approach is perhaps the most successful of these. In relating expert systems to the concept of logical sequential thought it is important to distinguish between different uses of the term "expert system." At one level the term merely describes a fancy bit of software the salesman would like you to buy, we will ignore this use of the term! Two other definitions are often implicitly used [for example see Beerel 1987 Ch. 5]:

a) a system which emulates a human expert

b) a system that combines a knowledge base and an inference engine to either solve problems directly or assist a human problem solver.

The first definition is more a statement of intent than of reality. The second definition is a more pragmatic one. It describes what most expert systems actually do.

There can be little doubt that expert systems were originally intended to replace experts and that they have proved very useful in many situations. Does this mean that they comply with both definitions? And why the mixture of success and failure in applying expert systems? It is often found in setting up an expert system that the knowledge is not easily obtained in the required form. The expert can perform the task but finds it difficult to state the rules that it is assumed he must be using. When rules are stated by an expert they are often inconsistent with his actual behaviour [Bereiter 1991]. An expert through a combination of basic ability, training and experience has developed an ability to perform certain tasks expertly, however the mechanism underlying that ability is a matter for speculation. The expertise has somehow become embedded in the expert. The use of rules and LST is one **untested** notion of how this may happen.

I suggest that what expert systems implement is logical sequential thought and that they are useful because they do something people find very difficult, that is to use LST. Three or four rules is all an average person can easily combine without assistance. This can be easily verified by the use of simple puzzles which state a small number of interrelated rules and ask for an inference based

on the rules supplied. Without recourse to pencil and paper most people find these problems very difficult and cannot handle more than a few rules at once.

It may be that expert systems are useful, not because they do what they were originally intended to do, that is, emulate experts, but because they do not. In the same way calculators are useful because they perform arithmetic which is something people are very bad at. The numerous failures in applying expert systems may be due to the fact that they do not work in the same way as human experts and hence they are appropriate for solving different types of problems. There is little evidence that human experts use rules for problem solving, except to speed up initial training [Bereiter 1991 ibid.], or to justify their actions later. The rule based performance of experts has always been an implicit assumption rather than a hypothesis which has been seriously tested. After all, how else could it be done?

There is even a special word in our language for the situations in which the expertise exists but it is difficult to see how the expertise could be explained in terms of rules and LST. We say it is a matter of **intuition**.

4.6 SUMMARY

The following points relating to logical sequential thought (LST) arise from the preceding analysis and speculation:

- It evolved only recently.
- It is easier to produce using language, it may even require language.
- It is not sufficient, by itself, for survival and too much may be bad.
- It is something people are not very good at but expert systems do well.

The key points from this chapter are;

- In trying to simulate the mental abilities of humans, researchers naturally focused on the ability that evolved last, the pinnacle of our evolution; logical sequential thought. This is not what enables us to survive, in fact it may be of little use without the other human abilities that evolved earlier.
- The reason we evolved the ability to use rules was as a way of training our neural networks without recourse to the real world. You have not "learnt" it until you stop using the rule.
- The frame problem does not need to be solved as such. The paradox of the stupid intelligent being who cannot look after itself is due to faulty assumptions about the interrelation between survival traits and definitions of intelligence. Once these assumptions are removed the "paradox" disappears.
- Expert systems are useful not because they do what experts do but because they do something we are very poor at, that is they use LST. (*i before e except after c provided the sound is e* is about the most complex rule set we can handle but a trivial task for an expert system).

4.7 FINAL AFTERTHOUGHTS [3]

- It might be argued that if a large artificial network was presented with a human scale training history it would simply clog with information. Alternatively, if the network was partitioned to avoid this, then maybe an analogue to the frame problem would arise; "Which networks should be brought to bear on which problems".

  Two points are worth considering here. Firstly, the speed of response of a network is not directly related to network complexity. In general the speed of response of a network will only depend on the number of layers and will be independent of the number of neurons and the number of connections. It is only the training of larger networks which may become problematic. This training problem is precisely the area which is tackled by the development of an educational environment, as described in later sections of the thesis. Secondly, while the brain is partitioned it is partitioned in a very special way. The hierarchical structure provides for the inhibition of primitive sub-networks by higher level, more complex systems [Scanlon & Johnson 1992]. This is done in a manner that ensures that the more complex networks can only overrule the lower, more primitive ones, if they have had time to react. An interesting analogy could be drawn between this approach and a successive approximation analogue-to-digital converter which produces an approximate answer quickly and more accurate results given more time.

- While conceding that intelligence is neither necessary nor sufficient for the survival of a creature, it might be felt that it is nevertheless very helpful. However, in the description of the frame problem by Dennett, intelligence is implicitly equated with the ability to survive [Dennett 1984 ibid.]. This confusion may be less important than the observation that intelligence

---

[3] These afterthoughts were inspired by discussion and subsequent private correspondence with Dr. Jack Copeland on a draft copy of this chapter [Copeland 1993]. However, the views expressed here are the authors and not necessarily those of Dr. Copeland. Dr. Copeland is a senior lecturer in philosophy at the University of Canterbury and author of the book "Artificial intelligence: a philosophical introduction" [Copeland 1993a].

should be considered as an adjunct to other abilities, rather than as a replacement for them.

- It could be argued that the frame problem itself does not require a solution if the design task is restated as "How can we use LST and yet avoid the frame problem?". We are an existence proof of at least a tolerable solution, however, determining the methods by which we solve this problem may be of considerable importance. The suggestion made here is that we avoid the frame problem by using LST as an adjunct to other methods and we do so only if time allows. The obvious ways of incorporating LST into artificial systems result in the frame problem, therefore the methods by which the advantages of LST can be gained without invoking the frame problem is an important area for further work. One other way in which LST could be utilised without producing the frame problem is to use it as an off-line pre-processor which produces extra inputs to the real time network system. Such extra inputs as are available will be used and those which have not had time to be processed will have no effect. Research is needed to determine how network inputs could be constructed to make use of such "please ignore - not yet available" values in addition to their normal input signal values and to determine how this would effect training.

- If the prime function of LST is not to solve problems then the question naturally arises as to what it is for. The suggestion made here is that it can be used "during periods of relative inactivity, to improve and develop skills which may be of significant use during survival activity." This can then be extended to the transfer of knowledge and skills from one person to another using LST, rules and language. This suggests that a key feature of biological intelligence is the development of an ability to transfer knowledge from one creature to another. This feature should, therefore, be involved in attempts to build artificial intelligent systems. The design and creation of an educational environment for the NNWF is a extension of this idea and is developed in the remaining chapters of the thesis.

4.8 CONCLUSION

The ideas presented in this chapter provide many possible avenues for future experimental research. One suggestion made is that a key feature of biological intelligence is the development of an ability to transfer knowledge from one creature to another. This feature should, therefore, be involved in attempts to build artificial intelligent systems. A possible role for LST in the knowledge transfer process has been suggested. The educational environment developed in subsequent chapters incorporate this use of LST together with other training heuristics. This environment is shown to improve the performance of the NNWF and extend the range of situations in which it can be used.

4.9 REFERENCES

page first cited

Beerel 1987
A.C. Beerel, Expert Systems strategic implications and applications, Ellis Horwood Ltd., Chichester England, 1987..... 4.11

Bereiter 1991
C. Bereiter, *Implications of Connectionism for Thinking about Rules*, Educational-Researcher, Vol. 20 No. 3, Apr. 1991, pp10-16..... 4.11

Boring 1923
E.G. Boring, Intelligence as the tests test it, The New Republic, 6 June 1923, 35-7..... 4.10

Card & Moore 1990
H.C. Card, W.R. Moore, *Silicon Models of Associative Learning in Aplysia*, Neural Networks, Vol. 3 No. 3, 1990, pp333-346..... 4.4

Collins & Smith 1988
A. Collins, E.E. Smith(Eds), Readings in Cognitive Science: A perspective from Psychology and Artificial Intelligence, Morgan Kaufmann, San Mateo, 1988..... 4.3

Copeland 1993
J. Copeland, senior lecturer in philosophy, Canterbury University N.Z., Private correspondence, Dec 1993..... 4.14

Copeland 1993a
J. Copeland, Artificial intelligence: a philosophical introduction, Blackwell, Oxford England, 1993..... 4.14

Dennett 1984
D.C. Dennett, *Cognitive Wheels: The Frame Problem of AI*, In Minds Machines and Evolution: Philosophical Studies, 1984, pp129-151..... 4.8

Dreyfus 1993
H.L. Dreyfus, What Computers Still Can't Do, MIT Press, Cambridge Massachusetts, 1993.... 4.9

Kohonen 1984
T. Kohonen, Self-Organization and Associative Memory, Third Edition, Springer Verlag, Heidelberg, 1984..... 4.4

Langton et al. 1991
C.G. Langton, C.E. Taylor, J.D. Farmer, S.R. Rasmussen, Proceedings of Artificial Life II , Addison Wesley, Redwood Calif., Feb. 1991..... 4.4

Manning 1979
A. Manning, An Introduction to Animal Behaviour, Edward Arnold Publ., London, 1979..... 4.4

Newell & Simon 1972
A. Newell, H.A. Simon, Human Problem Solving, Prentice Hall, Englewood Cliffs N.J., 1972... 4.3

Ornstein & Thompson 1986
R. Ornstein, R.F. Thompson, The Amazing Brain, Houghton Mifflin Company, Boston, 1986 4.10

Rumelhart & McCleland 1986
D.E. Rumelhart, J.T. McCleland, Parallel Distributed Processing, Explorations in the Microstructure of Cognition. Three Volume Set, MIT Press, Cambridge MA, 1986..... 4.4

Scanlon & Johnson 1992

R. Scanlon , M. Johnson, *How the Brain Works*, Technical Report from, Benet Laboratory, Watervliet Arsenal, Watervliet NY 12189-4050, 1992 ..... 4.14

Sternberg & Detterman 1986

R.J. Sternberg, D.K. Detterman, What is Intelligence? Contemporary Viewpoints on its Nature and Definition, Norwood, New Jersey, 1986..... 4.10

Sutton 1992

R.S. Sutton(Ed.), Reinforcement Learning, Kluwer Academic Publ., Boston, 1992..... 4.4

Chapter 5

5. An Educational Environment

This chapter introduces the idea of an educational environment for improving the performance of artificial neural networks. This initial training environment described here utilised four training heuristics. These were suggested by the speculative discussion of the previous chapter and consideration of techniques for assisting people to learn complex tasks. Three experiments are then described in which the performance of the neural network window filter was improved by the application of these heuristics. Later chapters formalize and extend the concept of educational environment for the neural network based window filter.

5.1 INTRODUCTION

During the experimentation with the NNWF it was found that for complicated tasks, starting the training for each task with a blank or randomly connected network would often led to slow learning or no learning at all. In natural systems each new task is not started with a blank or randomly connected network; sub-networks which have either evolved or learned to solve other problems are available to assist in the solution of each new task. By considering how a person could be helped to learn to perform complicated tasks and with the speculations of chapter four in mind, four training heuristics were proposed to assist with the training of artificial neural networks. In this chapter these four heuristics are introduced in the form in which they were originally proposed and an initial set of experiments using the heuristics is reported. The chapter essentially contains an extended form of a paper presented in 1993 [Pugmire et al. 1993a]. In later chapters a more formal basis for the heuristics is developed and the methods of implementing them are developed. The four heuristics are:

a. Learning simpler, similar tasks first.

b. Learning to perform useful sub-tasks.

c. Using rules provided by a teacher during training.

d. Structuring of the learning experience.

In the following section a brief description and rationale for each heuristic is given. This is followed by a short discussion of the importance of fuzzy targets for training networks. In section 5.4 a series of experiments are described which make use of these techniques to improve the performance of the NNWF. The chapter finishes with a discussion of these initial results and consideration of the further work which was required to formalize and extend the heuristics.

5.2 Rationale for the four Heuristics

In fitting an animal for survival, the ability to rapidly learn to perform a new task is of great importance. Simply as a result of previous experience, in the same environment, animals can make use of techniques implicit in heuristics **a** and **b**, learning simpler tasks first and learning useful sub-tasks. Only animals with some form of language can make use of rules in the manner envisioned by technique **c**. Any species in which parents, or other members of the herd, take some active part in the upbringing of young can make use of technique **d** for structuring the training environment. The four techniques are interrelated but to make the intention of the training procedures used in the following experiments clear and to provide some biological motivation each of the heuristics is described separately in more detail below.

5.2.1 Learning simpler similar tasks first

In teaching, one normally provides simple examples first. These simple examples should have some of the key characteristics of the final, more complicated, task to be learned. For example trainees might first learn to recognise clear pictures of a particular type of aircraft, later they could learn to recognise noisy or partially obscured pictures of the same aircraft. This learning of a simpler task first can dramatically improve the ease with which a task is learnt.

5.2.2 Learning to perform useful sub-tasks first.

In making distinctions between different objects people first learn to pick out the objects, their various parts and visual features. For example, a laboratory assistant might be taught to recognise the various parts of a cell. Later they will learn to recognise abnormal cells. This task is made easier by the initial mastery of the useful sub-tasks. Some of the visual features that may prove to be useful in defining sub-tasks are; textures, colours, edges and certain aspects of shading.

5.2.3 Using rules provided by a teacher during training.

While it is tempting to think of the use of rules and inference as a problem solving technique in itself, it is unlikely that this was the initial use of rules by the human race. We suggest that rules are principally used as an aid to learning

and it is in this sense that they are referred to here. The poor justification for the use of rules as a general problem solving technique is described by Bereiter [Bereiter 1991]. Thus a rule provides a method of transferring expertise and provides pre-training without the dangers of the real world.

There are two variations to the use of rules in this manner.

- A rule can be used to describe the approximate behavior required. As this relates to technique **a** where a simpler similar task is learnt first it will be referred to as technique **ca**.

- A rule can be used to provide examples of a useful sub-task. This relates to technique **b** for learning useful sub-tasks first and will be referred to as technique **cb**.

For example, if one wished to train a system to find the oranges in an image of an orange tree, then it might decided that an approximation to the required operation was embodied in the rule; *Find "round" blobs in the image.* This rule could be used to generate a number of contrived training images. Once this approximation to the required task had been learned we could further refine the operation by training with real images. This would be using a rule in the first way, referred to as technique **ca**.

Alternatively, if it was noticed that oranges always had a particular texture then a rule might be used to generate examples of texture with which to train a sub-network. This would be using a rule in the second way, referred to as technique **cb**.

The reason for training a network to perform the sub-task rather than implementing the rule algorithmically is to leave the system flexible and capable of further adaptation once the real data is presented. The rule may be only an approximation. The required behaviour will be embodied in the trained network.

5.2.4 Structuring the learning experience.

All teachers are aware of the importance of structuring the experience of students to make the required behaviour clear. An instructor wishing to teach a student to distinguish between different types of objects will carefully arrange the order of the examples presented. The instructor makes sure that the first examples are clearly different and do not contain unnecessary complications. If the student is having difficulty in making a particular distinction the teacher

provides additional examples that exemplify the distinction. For example, if a child who has mastered addition is being taught to multiply the order of examples must be chosen carefully. If the first example given is 2 x 2 = 4 the child may confuse multiplication with addition. This concept has been applied to the building of hierarchical networks [Whitcomb & Augusteijn 1991].

5.3 Fuzzy Targets For Training

Before moving to the description of the first set of experiments there is one other factor which is of major significance during training. As the pixel size in image processing systems is normally chosen to minimise the processing time while providing adequate resolution there will often be noticeable quantisation effects. This quantisation is particularly apparent in artificial images produced for training. During the course of this work the advantages of blurring the target for training became very clear. This can be related to the concepts of fuzzy logic [Hunt et al. 1992]. Alternatively, it can be seen merely as a method of compensating for the false precision introduced by using digital images. As the items to be identified in a real image are, in general, not aligned to the pixel grid, forcing the target to be at one pixel position rather than the next may cause faulty learning. The blurring of targets reduces this effect

The target images are also quantised in the intensity domain. Typically 8-bit values for each pixel are used. When a binary output is required, such as an edge position or object classification, it is usual to either threshold the network output or choose the largest of several competing outputs. The sigmoid output function used in back-propagation has an almost zero first derivative for values close to zero or one. As network weights are modified in proportion to the derivative this can lead to slow convergence for binary outputs. This is generally avoided by using target values which are slightly displaced from 0 and 1. Alternatively, a different error measure can be used which will cancel this derivative term in the weight update equation [Ooyen & Nienhuis 1992]. This was discussed in more detail in chapter three [see §3.4.3.1].

5.4 Experiments Using The Four Heuristics

In the descriptions of the experiments that follow, an emphasis is placed on the techniques for improving learning rather than the image processing tasks performed.

In order to investigate the usefulness of the four training heuristics a series of suitable tasks had to be found. This proved more difficult than originally anticipated as the tasks had to meet the following criteria;

1. The task must not be so simple that the NNWF could directly learn it without the use of the four techniques described above,

2. The task must only require the use of local information for its solution as a window filter is a local operator,

3. The task must be possible,

4. The task must use images for both input and output.

A human subject was used to ensure that the task to be performed was possible. If the subject could perform the task it must be possible, however, it must also be local. It is often not apparent that people are using global (or whole image) information to perform a task. To eliminate this possibility the subjects view of the test image was confined to a small aperture with the same extent as the input to the window filter.

As the output from a window filter is an image we have to carefully consider how to apply the NNWF to various image processing tasks. A typical window filter accepts an NxM image as an input and in response generates an NxM image as output, ignoring for the moment loss around the perimeter due to window size. When used for tasks such as noise reduction or edge enhancement an image is a suitable and obvious output form. When used for tasks such as object detection then some means of representing the results has to be incorporated into the output and target images. Possible representations include; a point at the centre of the object, a blob centred on the object, a blob covering the area containing the object, a duplicate of the input image in the area of the object or an ideal version of the object in the position of the found object. The choice of

an appropriate representation may be of major significance. Only a few simple representations were investigated in these initial experiments.

The capture and manipulation of images for these experiments was performed using NIH Image [Rasband 1995] on an Apple Macintosh computer with additional user routines written in Think Pascal and Think C. The neural network simulations were performed on a Decstation 3100 computer using the Aspirin/Migraines package [Leighton 1991] with additions written in C to process data files containing images.

5.4.1 Task 1: Improving the Marr-Hildreth emulation

The first task which the NNWF had difficulty with was the emulation of the Marr-Hildreth type edge detector. The experiments reported in chapter three showed a final RMS error of 47.1 for the full microscope image. Using the idea of learning a useful sub-task first and with the knowledge that this filter is normally implemented as a two step sequence the following experiment was devised;

Several small networks would be trained to perform the key parts of the MARHIL operation and then combined for fine tuning. The particular MARHIL operator being emulated is discussed in chapters two and three, it first convolves the image with a Laplacian of a Gaussian and then looks for zero crossings. The zero crossings are detected by comparing each pixel position, in the convolved image, with the pixels to the right and below. For this to be done in a single pass it is necessary to compute convolutions with three different offsets (0,0 0,1 & 1,0) and then perform a three input XOR on the results.

Three small networks were trained to perform the convolution each with a different pixel offset for the output. The targets for training these networks were produced by performing the convolution and then introducing an offset in the output image. The networks used had the following characteristics;

9x9 input window

no hidden neurons

1 output neuron

A dynamic learning rate was used and each network was trained for 1,000 epochs. Final RMS error rates were 14, 12, and 11. The weight values for the three networks are shown below in figure 5.1 as greyscale images. The bias level

is shown in the top bar and mid grey represents zero. The offsets of the centre point can be clearly seen in b & c.

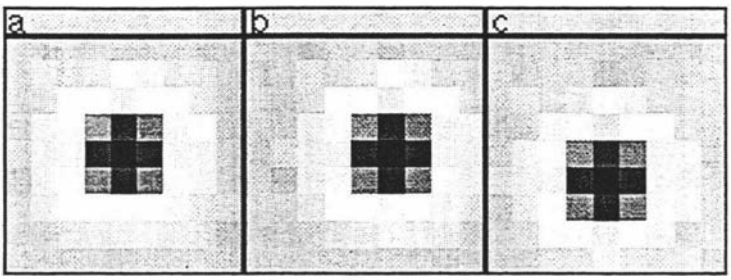


Figure 5.1: Weight values for sub networks

A fourth network was trained to perform the three input XOR function. The training set for this network consisted of the eight possible binary input patterns with the correct output as a target. The network had three hidden nodes and one output node. Again dynamic weight update was used. The final NMSE error was 0.00018 after 1000 training epochs.

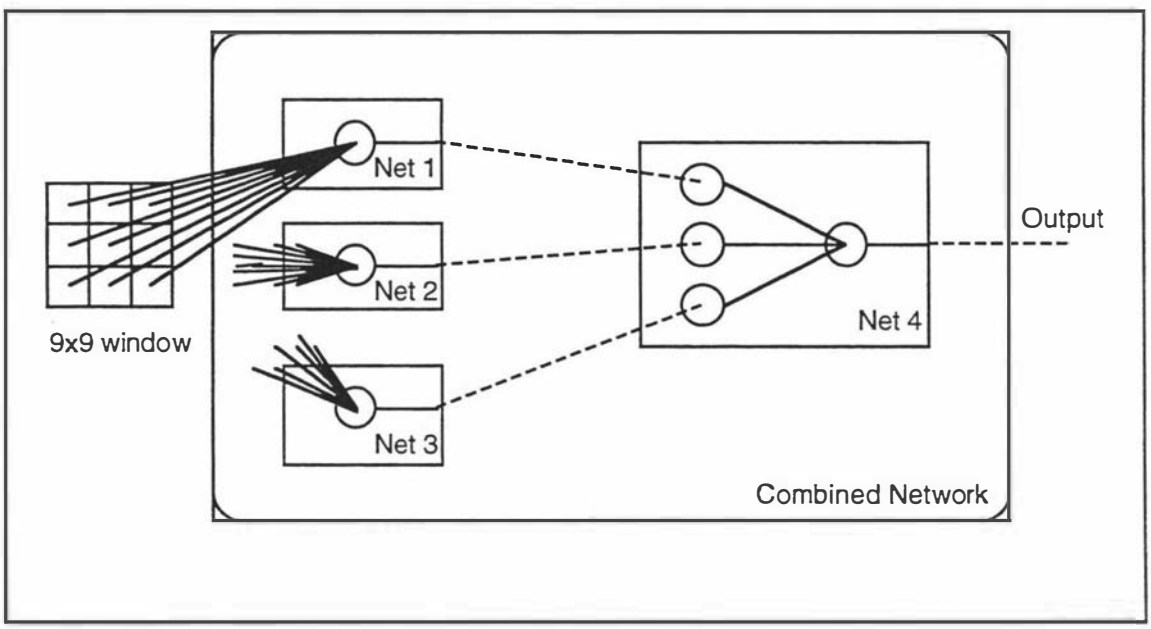


Figure 5.2: Combined network structure

The four networks were then combined into a single network as shown in figure 5.2 and loaded with the weights from the pre-training. This network was trained for a further 100 epochs and then its performance was tested on the two full images. The improved performance is shown in figure 5.3 in which the output from the MARHIL operator is shown on the left and the output from the trained NNWF is shown on the right.

To ensure that the new network structure was not responsible for the improved performance. The same network was also trained starting with random initial weights. It was then used to process a complete image and the output image was compared with that produced by the prototype MARHIL operator. The results of this test were then compared with the original results and the results reported above. The composite network was also trained using 0.1, and 0.9 target values rather than the 0 and 1 values used in the other experiments. The comparative results are shown in table 5.1 below.

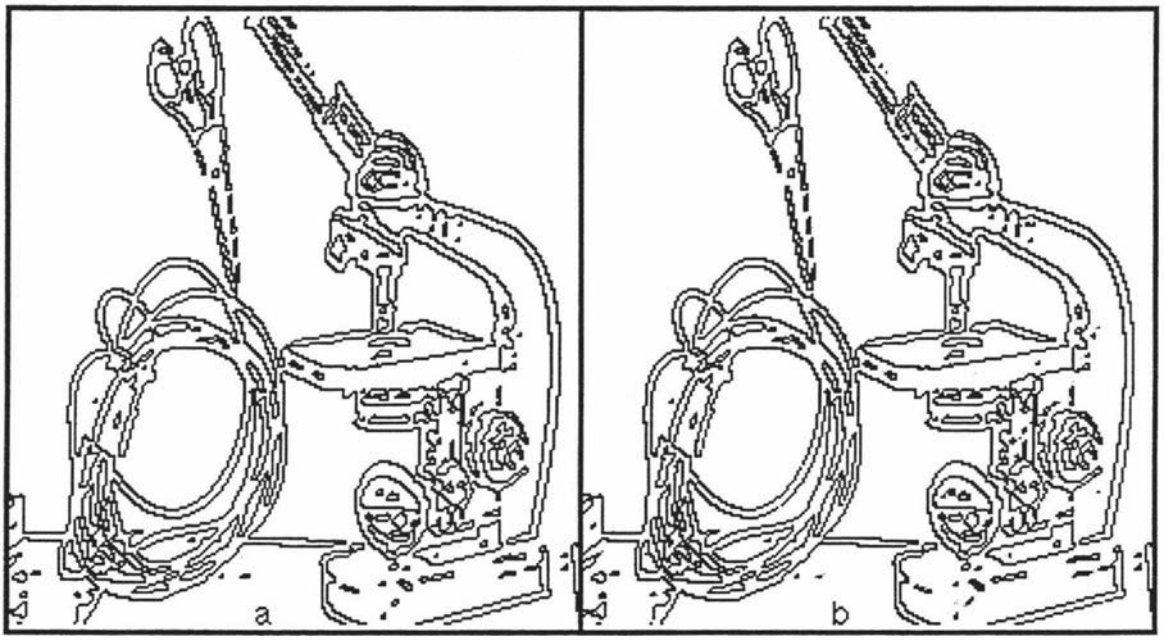


Figure 5.3: Output of combined NNWF

Network Type and Training Method	RMS Error
Original NNWF	47.1
Composite NNWF without pre-training	37.1
Composite NNWF with pre-training	16.9
Composite NNWF with pre-training (with 0.1 and 0.9 target values)	16.9

Table 5.1 Comparative Results for Marr-Hildreth Emulation

These results show a clear improvement in the learning performance of the NNWF when trained in this way. The modified target values did not produce any further improvement in performance.

5.4.2 Task 2: Finding E's in images of printed text

The second task investigated was that of locating occurrences of the letter **e** in an image of printed text. The images chosen were low resolution gray scale images of part of a newsletter. The images were intentionally captured almost at the limit of the resolution required for reliable identification of letters and the scene lighting was uneven. This provided a relatively difficult task. A section of one of the images used is shown in figure 5.4 with the extent of the filter window indicated by a black square. In figure 5.5 an area of the image surrounding a single word has been magnified to show the pixel resolution. The task is to identify the letter **e** everywhere in this text. Three possible forms of target and filter outputs were investigated:

1. A blob covering the entire letter at the position of each **e**.
2. A small dot at the centre of each **e.**
3. An image of an **e** at the location of each **e**.

For the third of these either a "perfect" **e** or the actual **e** image in that position could be used, so that the filter was gating the input image. A previous experiment had shown that with computer generated text this was a trivial task. However initial tests with this rather poor image gave completely negative results suggesting the task was sufficiently difficult to warrant further investigation. The NNWF used for this experiment had a window size of 15x15 pixels and contained two hidden layers, the first containing five neurons and the second three neurons.

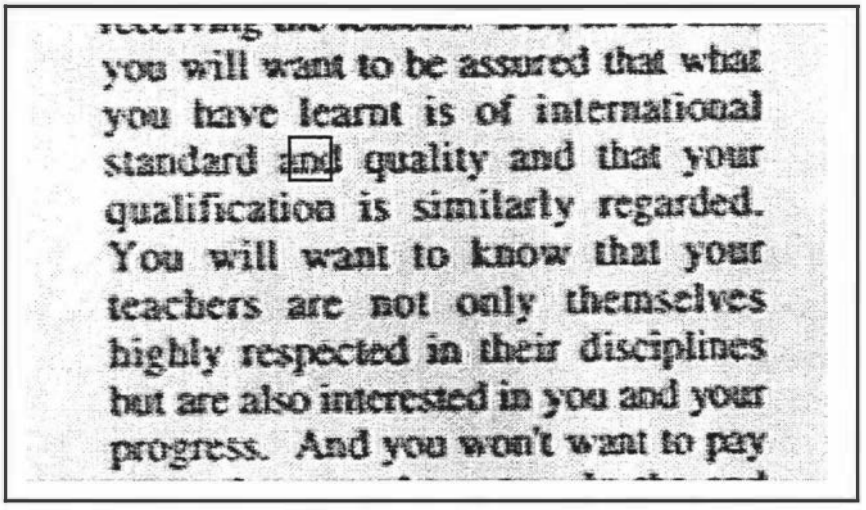


Figure 5.4: Image of a section of a newsletter
The extent of input to the filter is shown by the black square.

Experiment 2.1 (single dots as targets)

Training was performed on the left half of an image and testing on the right half. The output of the neural network was thresholded to provide a classification. In an actual character recognition system outputs would be provided for each letter and the most active unit would provide the classification.

As a measure of performance we counted the number of **e**'s correctly marked (correct positives cp), the number of non **e**'s marked as **e**'s (false positives fp) and the total number of **e**'s. Where a single figure of merit was required these were combined to give a generalised performance measure.

$$GP = (cp - fp)/(\text{number of e's}) \times 100\% \quad \textbf{5.1}$$

In order to apply this measure the greyscale images produced by the NNWF must first be thresholded. Using just a dot at the centre of each **e** in the training image produced very few correct classifications in the test image and even with careful selection of the threshold level, the GP figure was still negative.

Experiment 2.2 (fuzzy targets)

The first step towards enabling the network to learn this task, was to realise that a target representing a rough probability that each pixel position was the centre of an **e**, was more helpful than a single dot. This can be related to the concept of fuzzy logic [Hunt et al. 1992 ibid.] or simply considered as a method of

combating the unnatural spatial accuracy introduced by using digital pixel based images. An image consisting of blobs placed by hand over each **e** was skeletonised and blurred to produce this fuzzy target image. After training with this new target the NNWF was able to correctly identify 80 of the 107 **e**'s in the test image and produced four false positives. This gives a GP figure of 71%.

Examination of the NNWF output showed that the main source of error was the confusion of c's and e's. The similarity of these two letters can be seen clearly in figure 5.5 below.

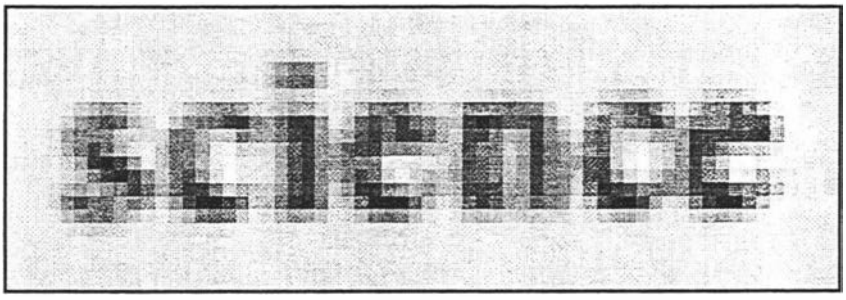


Figure 5.5: Single word enlarged to show effects of digitisation

Experiment 2.3 (structured training)

In this experiment the presentation of examples from the training set was carefully controlled. The NNWF training procedure was modified to use only those positions in the image at which either there was a target or the partially trained net from the first experiment produced some output. Some background areas were also included by blurring the image of targets or partial responses and using this as a mask. This is an example of a structured learning experience. If a child is having trouble distinguishing horses from cows a teacher might show them an edited series of examples to make the distinction clear. The use of this technique increased the number of **e**'s correctly classified to 102 with four false positives. This result gives a GP figure of 92%.

Experiment 2.4 (pre-training on a simpler similar task)

As an experiment on pre-training on a simpler similar task the NNWF was first trained on a computer generated image of text and then moved to training on the real data using the smoothed target image. The computer generated text was blurred 17pt Times. This was chosen as similar in size and font style to that seen in the captured image used in the previous experiments. This task is simpler because all the **e**'s generated in this way are identical. The NNWF quickly learnt

to perform this task with 100% accuracy on the test half of the computer generated text image. The training was then continued on the "real data". This produced a correct classification of 105 **e**'s with 14 false positives. This result gives a GP figure of 85%.

Experiment 2.5 (using a rule to define a sub-task)

This experiment made use of a rule to define a useful sub task or hint. If a vertical line is passed through an **e** and the intensity along the line is plotted there will, in general, be three peaks. This is not true for all the **e**'s in the data set and it is true in many other parts of the image. However it was hoped this could provide a useful hint or sub-task. A routine was added to the NIH Image package which marks such positions in an image. These processed images could be used both as a hint and to pre-train a sub-net.

Initial attempts to train a sub-net to perform this sub-task failed, so the "three peaks" images were used as a second input image which was scanned by extra inputs to the NNWF. This technique correctly identified 97 **e**'s with 37 false positives. This gives a GP figure of 56% which is worse than results from experiment 2.2. This presumably indicates that this sub-task does not transform the input data into a useful form, that is, the sub-task only processes data whose relevance is easily learnt from the training data.

Experiments 2.6 - 2.8 Combining techniques:

The techniques used above were then combined to see if this would produce a system with even better performance. The performance for each of the techniques separately and the combinations are shown in table 5.2.

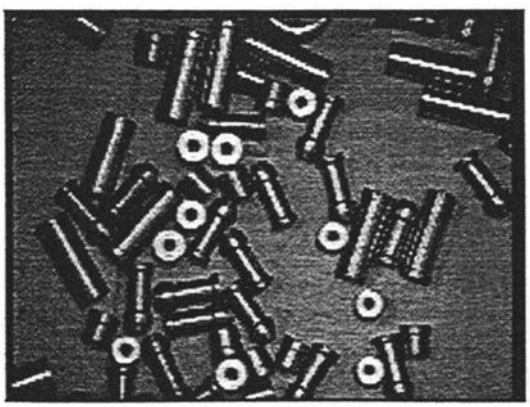
None of the combinations provided a performance figure better than the structured training technique alone although one combination found all 107 **e**'s but at the cost of more false positives being generated. Further investigation of when and how the heuristics should be combined is needed. This is discussed further in chapter seven [§7.5].

Experiment	Technique	nc	fp	GP	Comments
2.2		80	4	71%	blurred target
2.3	d	102	4	92%	structured training
2.4	a	105	14	85%	computer generated **e**'s
2.5	cb	97	37	56%	three peaks rule
2.6	a d	107	34	68%	2.4 and 2.3 combined
2.7	cb d	74	33	38%	2.5 and 2.3 combined
2.8	a cb	96	99	-3%	2.4 and 2.5 combined

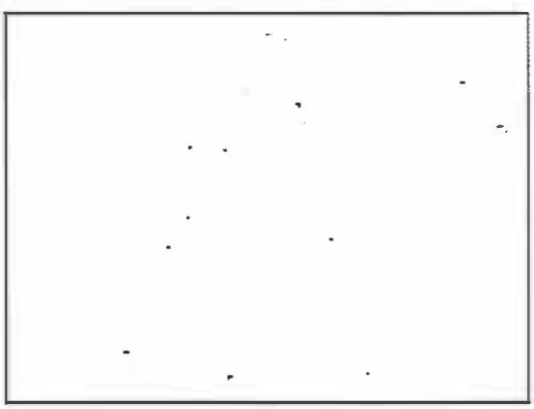
Table 5.2: Comparative results

5.4.3 Task 3: Identification of valve parts

The next task was the identification of valve parts. The image in figure 5.6a is of a collection of machined valve parts. The image processing task is to mark the centre of each of the round doughnut shaped pieces. The difficulty in this task is a result of the relatively small number of targets. There are only ten such objects in a 192 x 144 image. While the task can be described quite easily there are only ten correctly classified examples available for use in training.



a) Input Image



b) NNWF Output Image

Figure 5.6: Identification of machined parts

Experiment 3.1

In the first experiment the centres were marked by hand. These marks were then used to train the NNWF. The network failed to learn to perform this task with either a single dot at the centre of each object as a target or with a blurred version of this to allow for inaccuracies in marking the centres.

Experiment 3.2

As the task can be defined quite easily the use of a rule to generate extra data for pre-training was appropriate. The simple rule that the correct blobs have both a light periphery and a dark centre was used to generate the training input and target images shown in figure 5.7. These were used to pre-train the network. After 39 epochs the network was down to 5 errors on the pre-training data. This trained net was then used to process the real data. With a threshold set to 127 this gave 9 correct marks and 7 false positives. By choosing a lower threshold this could be improved to six correct and no false positives.

Experiment 3.3

In the next experiment the NNWF was pre-trained on the generated data and then training was continued on the real data. After 50 epochs of training on half of the real data the UWF gave 9 correct marks and 1 false positive with a threshold set at 127. The un-thresholded output of the network obtained following this procedure is shown in figure 5.6b. It can be seen in this image that before thresholding three other false positives existed. It is also interesting to note that the one piece not found (lower right of image) was an isolated part. There were no isolated parts in the training half of the image.

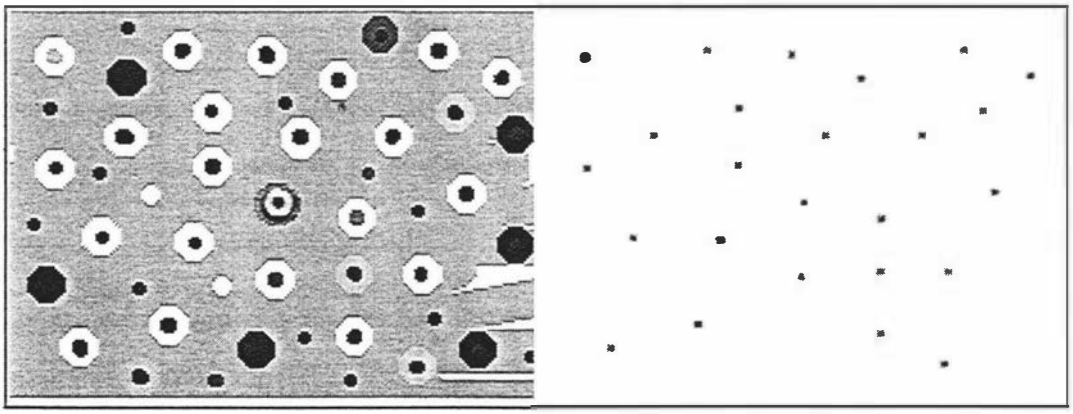


Figure 5.7: Rule generated input and target data

Here, by the introduction of pre-training on a simpler but similar task defined by a rule, it has been demonstrated that the performance of a network can be changed from being totally inadequate to most satisfactory.

5.5 DISCUSSION

These preliminary results showed definite benefits from the use of the four heuristics for improving the effectiveness of training. Further work needs to be done to determine the characteristics of useful sub-tasks. A key feature here may be the extent to which sub-features of an image are expressed in a particular task. For example if a particular form of texture is an important guide but the concept of that texture form itself is not easily discernible from the particular task then this should be a useful sub-task to learn.

The results of experiment two compare favorably with the performance of optical character recognition systems [OCRs]. However two things should be noted here, one which makes the task easier than the normal OCR task and one which makes it more difficult; The first point is that a single font and character size is being used both during training and testing. The second point is that a low resolution grey scale image is being used which would not normally be considered adequate for input to an OCR system.

The methods of combining networks, some of which may have been pre-trained to perform sub-tasks, also needed further investigation. In these experiments the sub-tasks were handled algorithmically and only provided extra inputs or hints. The use of sub-networks to perform these sub-tasks should provide greater flexibility.

The use of hints forms one part of a training environment. Abu-Mostafa has shown that a hint only needs to be able to generate an error signal, and not a target value, to be incorporated into back-propagation learning [Abu-Mostafa 1993]. This should allow additional types of hints to be incorporated into the training environment. Both of these points are considered further in later chapters.

5.6 REFERENCES

page first cited

Abu-Mostafa 1993

Y.S. Abu-Mostafa, *Learning from Hints*, in Proceedings of Third Australian Conference on Neural Networks ACNN'93, Melbourne, Feb 1993, pp37-40 ..... 5.18

Bereiter 1991

C. Bereiter, *Implications of Connectionism for Thinking about Rules*, Educational-Researcher, Vol. 20 No.3, Apr 1991, pp10-16 ..... 5.4

Hunt et al. 1992

B.R. Hunt, Y.Y. Qi, D. DeKruger, *A generalization Method for Back-Propagation using Fuzzy Sets*, in Proceedings of Third Australian Conference on Neural Networks ACNN'92, Canberra, Feb 1992, pp12-16 ..... 5.6

Leighton 1991

R. R. Leighton, The Aspirin/MIGRAINES Software Tools: User's Manual 5.8

Ooyen & Nienhuis 1992

A. Van Ooyen, B. Nienhuis, *Improving the Convergence of the Back-Propagation Algorithm*, Neural Networks, Vol. 5, No. 3, 1992, pp459-463 ..... 5.6

Pugmire et al. 1993a

R.H. Pugmire, R.M. Hodgson, R.I. Chaplin, Teaching a Neural Network based Window Filter to Perform Difficult Tasks, Image and Vision Computing NZ '93, Auckland, 1993 ..... 5.2

Rasband 1995

W. Rasband, NIH Image: An image processing software package for the Macintosh, 1995, available free of charge from Wayne Rasband over 5.8

Whitcomb & Augusteijn 1991

M.J. Whitcomb, M.F. Augusteijn, *Hierarchical Learning in Neural Nets - A New Paradigm*, in Intelligent Engineering Systems Through Artificial Neural Networks, ASME Press1991, pp106-112 ..... 5.5

Chapter 6

6. Instructional Design and Training Artificial Neural Networks

In the previous chapter four heuristics for improving the training of artificial neural networks were presented together with examples of their use. In this chapter, in order to develop an improved method for training artificial neural networks, the methods used to teach people are considered in detail. The general field of study is that of education. In this thesis the focus is on methods of improving learning performance for a given task or skill by a given pupil. The specific area in the field of education that considers this question is that of instructional design(ID). This chapter discusses instructional design and relates it to the training of artificial neural networks(ANNs). The use of this analogy has been found helpful in formalising the design of an educational environment for training artificial neural networks.

There are three aims for this chapter:

- To place the heuristics identified in chapter five in a context of instructional design theory. The question to be considered is; "Do the heuristics suggested for training ANNs have counterparts in standard theories of instructional design for people?"

- To seek further useful ideas or strategy components from instructional design theory. The question asked is; "Are there further lessons to be learnt from the analogy of teaching people, which are not covered by the heuristics already proposed?"

- To provide some initial foundations for a theory of instructional design for artificial neural networks.

There may also be insights from the field of neural networks that could be transferred back to instructional design for people. This was not a focus of this work.

It should be noted however that while a general ID theory would apply to all trainable ANNs the work discussed in this thesis has focused on training a back propagation network to perform as a universal window filter in image processing tasks. Much of what has been learnt is relevant to other ANN work but further research will be required to determine how generally the results can be applied.

To fulfill the aims listed above it is first necessary to consider the nature and function of instructional design as applied to people. This is reported in section one of the chapter which follows the analysis of Reigeluth developed in "Instructional Design Theories and Models" [Reigeluth 1983a]. Next, the difference between instructional design for people and instructional design for ANNs is considered. In section three, several specific theories of instructional design are critically examined. In this discussion comments are made on both proposed heuristics and new heuristics. In section four the key points identified during the examination of ID theories are summarised and discussed. In the last section of the chapter the question of what should be expected of a theory of instructional design for ANNs is considered. Some tentative principles of instructional design for ANNs are then proposed. Finally comments are made on the additional research still required in this area.

6.1 What is Instructional Design

"Instructional Design" or "Design Science" is that part of the general field of education which is concerned with prescribing optimal methods of instruction to bring about desired changes in knowledge and skills for a given set of students. It is clearly related to learning theory but has a different focus. Instructional design focuses on the entire environment in which the pupil is placed and how this will effect his learning. This is not to deny the importance of the learner but merely to focus on creating the best learning environment and then to look at the learner-environment interaction from this point of view. Briefly, instructional design deals with how best to teach a given course to given students and tries to provide a blueprint of what methods to use for particular desired outcomes and students.

Reigeluth suggests the following framework for use when evaluating theories of instructional design;

- Methods — Relevant variables that can be controlled.
- Conditions — Relevant variables that are fixed.
- Outcomes — Results of the application of particular methods in particular conditions

A particular theory of ID must consider these three general aspects and provide a coherent set of principles for choosing appropriate methods for given conditions and desired outcomes.

Methods can be broken down into three categories;

- Organisational strategies — Elemental methods of organising subject matter such as the use of examples or diagrams and sequences of the selected content.
- Delivery strategies — Such as the choice of teachers and media and their characteristics.

- Management strategies — Which are methods of choosing or sequencing the organisational and delivery strategies.

The organisational strategies can be further partitioned into either micro strategies, which deal with a single idea or macro strategies, which deal with the presentation of more than one idea.

An instructional model consists of a complete set of strategy components and their relation to conditions and outcomes. A prescriptive model or theory will prescribe the best methods for a given set of conditions and desired outcomes. A prescriptive theory would thus contain a number of prescriptive principles of instruction such as;

> *To increase long term retention, begin instruction with an overview that epitomizes the content rather than summarises it. Then gradually elaborate on each aspect of the overview, one level at a time, constantly relating each elaboration to the overview.*

A very useful set of criteria by which instructional theories may be judged is suggested by Reigeluth [Reigeluth 1983a ibid. p25] and reproduced here with minor modifications;

- Theory — Is it a theory or just a list, model or set of beliefs?
- Consistent — Is it internally consistent?
- Limitations — Are its boundaries and limitations clearly defined?
- Valid — Is it contradicted by any known data?
- Simplicity — Is it unnecessarily complex? (Like modern physics)
- Usefulness — Does it provide useful predictions or assistance in design?
- Comprehensive — How much of the total variance is covered?
- Optimal — Is there anything better?

These criteria provide an excellent yardstick for evaluating a theory of ID or any other scientific theory. Theory construction is, however, an iterative process and many of the present theories of ID do not satisfy the criteria suggested above. A common fault appears to be the simple classification of conditions or methods described as part of a particular model are actually found to be unsuitable when

principles relating conditions to methods are determined. The simple and tidy classification of conditions is then modified or extended in an ad hoc manner.

Following the next section, which considers the difference between instructional design for people and instructional design for artificial neural networks, a number of ID theories are examined in detail. For this analysis, in addition to the criteria above, each theory is considered in terms of the three aims of this chapter.

- Does it provide a context for the heuristics proposed in chapter five?
- Does it provide any additional heuristics for training artificial neural networks?
- Does it assist in laying foundations for a theory of instructional design for ANNs?

It is important not to be too literal in looking for analogies. Each concept or principle used in an ID theory needs to be considered in relation to training ANNs. Is there an obvious analogy? If not, is there some facet of ANNs which is loosely analogous? It is well to be wary of deciding too soon that there is no analogy for a particular aspect just because ANNs are so much simpler than people. For example, the concept of motivation or the use of cues might be discarded as irrelevant to ANNs but with further thought it may be possible to relate these ideas to the gain term in a learning algorithm or to pre-trained sub networks of a special type. Once the analogy has been found then simplified versions of the instructional design principle or model aspect under consideration may be found quite useful in the training of ANNs.

6.2 Differences between ID for people and ID for ANNs

Naturally there are differences between instructional design for people and instructional design for artificial neural networks. These differences arise in several ways.

The human student is a relatively fixed entity, the underlying mechanisms governing learning cannot readily be changed by the teacher. When using ANNs the learner or pupil is not fixed. In fact the majority of work on training ANNs has been focused on improving "the pupil" with scant reference to instructional design [Ooyen & Nienhuis 1992, Møller 1993, Riedmiller & Braun 1993, Tollenaere 1990, Fahlman & Lebierer 1990]. It is not suggested that this is not worth doing but only that it is but one aspect of the problem and work on other aspects is also worthwhile.

Secondly, in training people a teacher has the option of stepping outside the actual training exercise to help the learner who is having difficulty. This ability to use verbal strategies to rapidly change tack can play a significant role in teaching. For example a teacher might say;

> "What I am trying to get across is that the sort of movement required here is...
>
> and perhaps if you tried doing X it might help you learn this task more quickly"

In ID for ANNs this can only be done in much more limited ways. Suggestions of different ways the pupil might behave have to be translated into changes in the actual learning algorithm. Any appeal to the learner to approach learning this particular task in a special way has to be translated into changes in examples, presentation order, alternative error calculations, structural modifications to the network or some other explicit feature of the neural network or its training environment.

Before discussing ID further it is perhaps appropriate to comment on the present trend in educational research to focus on assisting the pupil to learn rather than teaching [Entwistle & Ramsden 1983]. This shift in focus was probably necessary after a long history of educational research on teaching, which with

time led an over emphasis on the role of the teacher as the active participant. The learner is the one in which it is hoped the change will take place and hence it should be the learner who is the major focus during the learning process. In the study of artificial neural networks no such over emphasis on the teacher has occurred. In training ANNs the main focus has been on modifying the pupil to make it better at learning, an option not normally available in human teaching! For this reason in this thesis an emphasis is placed on the training environment for ANNs, that is, on teaching.

The following definition of teaching has been used;

> Teaching is the process of arranging the form of presentation, variety of material, order of examples and any other relevant aspects of the learning environment in such a way as to make it as easy as possible for the learner to learn.

This differs from the standard dictionary definition, *to enable or cause to do*, by stressing the importance of enabling in contrast to causing. It also defines and limits the type of methods to be considered in this work.

Another major difference between human and machine learning, is that, in most training situations involving people, the instructor knows how the problem should be solved. It is merely a matter of transferring the ability from the teacher to the student. For example, if the task is to solve simultaneous equations, the instructor knows how this can be done. The desired outcome is that at the end of the training the student will be able to solve simultaneous equations using the same method. In other tasks the solution method may be unknown or only approximately known, for example teaching someone to play a particular shot in table tennis. In these cases the teacher is trying to assist the student to discover a solution method and hence gain the desired skill. The required net result is known but not the method of obtaining it. This second situation is more akin to training artificial neural networks. Where an explicit solution method or algorithm is known, other non neural network approaches are generally preferable. This is not always the case; a known algorithmic method may be too slow or a NN solution may be used to provide an inherently parallel implementation.

Another difference to keep in mind is that ANNs can be returned to an initial or earlier state. In fact the standard approach in training ANNs is to revert to a new set of random weights at the beginning of each new task. This is not possible in

human training, we can never go back. For this reason it is important, with people, to ensure that incorrect responses are never practiced. They will have to be unlearned later. This may not be so important in training ANNs as a set of weights can be saved and later restored, however if the training of ANNs is to be made more like human training this may involve the loss of the capability to reload weights. If this should turn out to be the case then it will become important to ensure that ANNs do not practice incorrect responses during training.

Lastly, the ANNs available at the present time are relatively limited in their capabilities. This will sometimes restrict our interest to a subset of the components of a particular ID theory. For example in the forthcoming discussion of Component Display Theory only a subset of the performance-content matrix proposed in that theory is relevant to training present ANNs. Therefore the parts of the theory relevant to that subset will be discussed in more detail than the rest of the theory.

6.3 THEORIES OF ID AND THEIR RELEVANCE TO ANNS

6.3.1 Early Theories of ID (Gagné & Briggs)

The work of Robert Gagné and Leslie Briggs has been of major significance in answering the question; What is known about human learning that is relevant to the design of instruction and how can this knowledge be organised in a systematic way to assist in the design of instruction? For this reason some of the early work by Gagné and Briggs is considered first in this section on theories of instructional design [Aronson & Briggs 1983, Gagné & Briggs 1974, Briggs 1977].

Gagné has classified human learning into five categories with different internal and external conditions hypothesised as being necessary for learning to take place in each category. To translate these terms into the framework suggested in section 6.1 the five categories indicate the conditions that are present. The internal and external conditions hypothesised as being necessary for learning to take place are a description of methods. As these two descriptions involve different uses of the word **conditions,** the nomenclature of Gagné and Briggs is used during the description of their work. The five categories suggested by Gagné are;

- intellectual skill
- motor skill
- verbal information
- cognitive strategy
- attitude

The first two are the most relevant to the training of ANNs. The third category relates to memory either directly addressable or content addressable. It also inherently involves the use of language and is therefore not discussed here. The cognitive strategy category represents a level of learner control and generality that is not envisaged in our present ANN systems. The neural networks of today are not yet sufficiently general purpose that a learnt cognitive strategy would

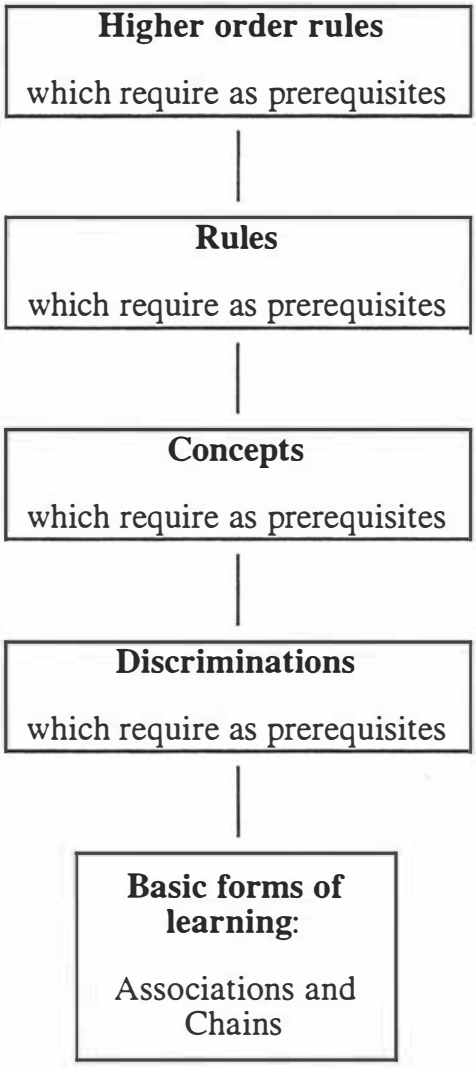
be useful. The learning of attitudes is also beyond current ANN capabilities. The prejudices of artificial neural networks are not yet of great social significance!

This leaves the two categories of major interest; learning motor skills and learning intellectual skills. The first of these is of particular relevance to neural networks being used in robotics but may also have relevance to the type of neural networks discussed in this thesis. The skills required of the NNWF while not motor skills, do have some similarity in that the exact solution method is generally not known by the teacher. Only the desired overall behaviour is known.

The **external conditions** suggested for the two categories of interest are; the **presentation of examples** to classify (and receive feedback on) and the opportunity to **practice** and receive **feedback**. These are both clearly available in the standard methods of training ANNs.

The **internal conditions** postulated are the **recall** of certain previously learned **prerequisites**. This idea of prerequisites can be related to the sub-tasks heuristic suggested in chapter five. The identification of prerequisites or sub-tasks, however, has proved much more difficult in our work on ANNs than might have been expected [see §5.4.2 Exp. 2.5]. In many cases the learning of what appeared to be a useful sub-task did little or nothing to improve the learning of the complete task.

Gagné suggests a sub-classification of intellectual skills with each level in this classification scheme using the previous level as a prerequisite. This hierarchy is as follows;



This classification appears to involve the assumption; that the only way of using a concept is by application of a rule. This assumption is found in much of the education literature and is related to the assumption that *experts must use rules* which is discussed in chapter four.

Consider a situation in which a network is trained to respond to pictures of chairs in one way and pictures of tables in another. Let us suppose that the performance of a trained network was found to match the responses of people given the same task. Let us further suppose that the performance of systems using rule based definitions of a chair and a table could not match the human

responses as closely. In such a case it would seem reasonable to consider the network (either the artificial one or the human one) as a definition of the concepts involved and the rules merely as an approximation to the concept. That the reverse is usually done may well merely reflect the lack of experience the educationalists have with artificial neural networks.

The remaining part of the classification hierarchy, however, suggesting discrimination and sub-concepts as being essential sub-tasks, supports the sub-task heuristic introduced in chapter five. It also suggests likely forms for these sub-tasks;

- Manipulations of the input data that will help in discrimination.
- The use of sub-concepts that could be used in a rule based description of the higher level concept.

Note, however, that a sub concept that is used in one form of a concept definition or implementation may not assist learning of this concept if a different statement or functional description of the concept is finally acquired by the learner. For example consider a NN system for recognition of images of New Zealand city names on envelopes. A learning hierarchy may identify recognition of individual characters as a learning pre-requisite. This presupposes a method of solution (i.e. identifying letters and then considering groups of letters), however, it may be that the city names can be learnt more efficiently as complete units. For example a particular implementation of the concept, **this is an image of the name for Auckland city,** may not involve the sub concept, **this is an image of the letter n**.

In addition to identifying what should be taught in terms of a learning hierarchy of prerequisites, Gagné's work also suggests a sequence of learning to apply when prerequisites themselves have prerequisites. The order of learning is from the bottom up while the order of use in performing the total task may quite different. Opportunities for using this sequencing in training the NNWF have not been found so far. This may be because the tasks studied were relatively simple. This type of sequence if taken to extremes produces a series of unrelated learning tasks with no structure in which to embed them during learning. This is discussed further in section 6.3.4 on elaboration theory.

6.3.2 A Behavioural Approach (Gropper)

The behavioural approach dominated learning theory and instructional design until the early 1970s. "A Behavioural Approach to Instructional Design" by Gropper is considered as a representative example of this approach [Gropper 1983].

In the behavioural approach the unit of analysis is the stimulus-response (S-R) association. Instructional Design in this context consists of providing the optimum conditions for the acquisition of the particular S-R pairs of interest. The criterion stimulus is the event that must gain control over the required response once learning has taken place. For example for a student learning to perform simple addition the criterion stimulus may be a sum of the form;

$$5 + 4 + 7 = ?$$

The required response would be to write the answer, **16**.

Learning requires that the student; **discriminate** between stimuli so that they only respond to examples of the criterion stimulus, **generalise** to other examples of the criterion stimulus not seen during training, **associate** the criterion with the required response and possibly **chain** several S-R steps together if the required behaviour involves a sequence.

A cue is a stimulus that elicits the required response before training. In the language of reinforcement, the criterion stimulus will become the conditioned stimulus(CS). The cue already elicits the required behaviour and is therefore referred to as an unconditioned stimulus(US).

The concepts of discrimination and generalisation have clear analogies in the training of ANNs, however, none of the heuristics introduced in chapter five appear to provide an analogy for the use of cues. This suggests we should ask how exactly a cue helps learning take place and whether the technique could be used with the NNWF.

In human and animal learning the cue can be seen as a way of ensuring that a correct response occurs so that it can then be reinforced. This is clearly relevant to reinforcement learning in artificial systems [Sutton 1992, Whitehead & Ballard 1990] but does not appear to provide any lessons for back propagation(BP) learning systems as used in the NNWF. An alternative view of the use of cues is possible.

Learning to perform a task can be thought of as the process of forming a linkage between an input (the criterion stimulus) and a required output (the required response). This linkage may take the form of a series of linked sub-tasks. From this viewpoint, a cue is the input to a sub-task near the end of the sequence. It is the input to a sub-task that has been learnt previously, maybe for some other purpose. At the start of training, presentation of the cue activates this final sub-task and hence produces the required output. It also provides a new target that is closer to the criterion stimulus. If cues are considered from this alternative point of view then they clearly relate to the useful sub-task heuristic. They also provide an extension to the heuristic suggesting that sub-tasks should be used to build inward from both the input and output end. This approach of using a solution graph to build from both ends has been found helpful in image processing algorithm development [Ngan 1993, Dorner 1983].

Gropper suggests three fundamental techniques are used to assist in the process of linking the criterion stimulus to the required response;

- Incrementing Learning a series or chain of S-R pairs.
- Shaping Learning a series of approximations.
- Fading Gradually weaning the learner from the use of cues.

Incrementing and fading both relate to the use of sub-tasks; incrementing uses a series of sub-tasks that together form a sequence from input to output. This series of sub-tasks may be learnt in any order. Fading uses a sub-task that is close, in terms of a solution graph, to the required output. The way in which the target value for the cue could be discarded in later stages of training would require further investigation, one possibility would be to slowly decrease the significance of errors on the cue output of the network.

Shaping consists of a learning sequence in which the performance criteria are gradually tightened. For example, if a pigeon is to be taught to press a food bar, any movement towards the bar would be rewarded in the early stages of training. The idea of shaping is thus analogous to the heuristic for learning simpler similar tasks first. It does, however, suggest a gradual change rather than the step change used in the experiments described so far.

In the behavioural approach to instructional design the following four topics are considered;

• Conditions	The things which make learning easy or difficult.
• Treatments (or methods)	The ways of assisting learning (teaching) such as different forms and sequences of practice.
• Principles	Matching treatments to conditions or micro strategies.
• Multiple Objectives	Macro strategies.

Each of these topics and its relevance to instructional design for ANNs is discussed below;

6.3.2.1 Conditions

The term conditions is used to refer to those factors that make a particular behaviour difficult or easy to learn. Task analysis is the process of identifying for a particular objective; what component skills are required, what are the stimulus response characteristics and what are the characteristics of the required performance level. The behaviourally defined component skills are discrimination, generalisation, association and chaining.

The following list of conditions affecting ease of learning component skills are identified by Gropper;

- similarity / dissimilarity of stimuli
- similarity / dissimilarity of responses
- number of classes of responses or stimuli
- number of properties on which class membership is based
- number of properties on which response membership is based
- length of chains
- strength of existing stimulus response associations.

The effect of each of these conditions can be stated as a series of behavioural principles relating to component skills. For example;

- The more similar the stimuli are that need to be discriminated the more difficult it will be.
- The more similar the stimuli that have to be generalised to the easier it will be.
- The more dissimilar the stimuli that fall into different classes the easier it will be.
- The more dissimilar the stimuli that fall into one class the harder it will be.

All of the conditions listed above also have significance for training artificial neural networks and may provide some useful guidelines as to which tasks will be more difficult for ANNs to learn. Several aspects of the relationship between these conditions and training neural networks are discussed next.

For ANNs the concept of similarity should be considered in terms of the resolution of the individual inputs and the number of inputs. Discrimination will certainly be limited by the precision of the input variables. The use of non-linear scales or sensors may assist in providing an extended range and high resolution in the most useful part of the range. Several workers have investigated the use of self organising networks to re-map the input in ways determined by the actual input data space [Kohonen 1990, Hrycej 1992]. For example Kohonen has shown that a self organising net can provide a remapping which is magnified in regions of the input space in which many events occur. The appropriate coding of the input data is also related to this condition of similarity or dissimilarity. The coding scheme chosen for an input variable can make particular pairs of data points either more or less similar. For example, time could be coded using two variables; one for day number and one for hours within the day, alternatively it could be coded using a single variable providing hours since some particular date. Which scheme is best will depend on how they help or hinder discrimination and generalisation which will in turn depend on the particular task.

The standard training approach for back-propagation networks is to start with a new set of random weights for each new problem. In this case the problem of the strengths of existing S-R response pairs does not arise. If pre-training or training with sequentially varying sets of input-target pairs is used this condition may become relevant. Robins and Ratcliff [Robins 1993, Ratcliff 1990] have investigated the problem of catastrophic forgetting in neural

networks. Catastrophic forgetting occurs when training on new examples causes previously learnt examples to be misclassified.

6.3.2.2 Treatments or Methods

At the heart of the behavioural instruction model is a requirement that students, or learners, must practice a designated response in the presence of the criterion stimulus that must come to control it. Therefore treatments are discussed entirely in terms of different forms and sequences of practice. The relevant variables, or treatment tools, listed below are identified by Gropper as the most important in producing different forms of practice. Each is discussed briefly in the paragraphs below.

- Degree of cueing.
- Size of the unit of behaviour practised.
- Mode of stimulus response.
- Variety of examples. (familiarity, similarity, saliency)
- Content of examples. (exaggeration of S-R examples)
- Frequency or repetition of practice task.

Cueing

The role of cues in practice is to help elicit the correct response so it can be reinforced. Cue strength should diminish during practice based on how good responses are. This implies that the network being trained is taking over the role of producing the cue or intermediate result itself. The process of producing the required output from this intermediate result already exists as the cue is an unconditioned stimulus (US).

Unit Size

The amount and fidelity of the required behaviour should increase during training. Thus at the beginning of training a small part of the behaviour will be practised and the required accuracy should be quite low. It is suggested that the degree to which the criterion behaviour is broken down during training should be based on the expected degree of difficulty. This could be restated using two of the heuristics introduced in chapter five as; try useful sub-tasks and simpler

similar tasks first. It may also be possible to incorporate these ideas as modifications to the error calculation. Minor errors should be considered less severe at the beginning of training. This would involve a modification to the standard BP error calculation which uses squared errors throughout training. One approach would be to decrease the power to which the error term is raised during the training process. To ensure the measure remained positive the unsigned error would have to be used. Another approach would be to provide training sets that included coarsely matched pairs. For example positive targets for **c**'s and **e**'s might be included in the training set (during the early stages of training) for a network which was to detect **e**'s.

Mode

The term *stimulus-response mode* is used to refer to two treatment techniques;

The first of these is the use of the following sequence as training progresses;

1. Initially try to recognise right and wrong responses,
2. Then try to correct near correct responses,
3. Then try to produce correct responses directly.

In the standard approach to training ANNs the output of the untrained or partially trained net is compared with a target value. The weights are then modified in a way which is dependent on this error or difference. In the early stages of training the responses produced by the network will be random and will in general include; correct responses, near correct responses and incorrect responses. As training continues the responses will be correct or nearly correct on an increasing percentage of practice examples. In this way the standard ANN training scheme has similarities to the sequence described above for training people. The network itself, however, is never presented with correct or nearly correct responses. A modification to the standard training procedure for ANNs could provide a more exact analogy as follows;

> Augment the network input vector with two or more vectors of the same size as the output vector. In addition to presenting the network with the normal inputs and outputs for each training example, present a correct or nearly correct output vector to one of the augmented inputs. Provide an incorrect output vector to the other augmented inputs. During a training sequence the augmented inputs would be moved from providing right

and wrong responses, through nearly correct responses, to no useful data and thence fading to zero. During this procedure it might be necessary to fix and then gradually fade weights from the augmented inputs.

The second technique covered under the heading of S-R mode is to use familiar concrete examples first and then lead on to abstract technical examples. For example, instruction could start with an example of a procedure, described in everyday terms, and then lead on to a formal technical description of the procedure. This is a form of the structured training heuristic introduced in chapter five.

Variety

Variety and order of examples are clearly of importance. A suitable variety of examples promotes generalisation and transfer. A suitable sequence of practice examples will also assist learning. The sequence is again related to the structured training heuristic. Ensuring the range of examples is adequate is a necessary part of training ANNs, a range of examples that is not representative of the actual population will usually lead to poor generalisation.

Content

Several variations in the content of examples will have significant effects;

- Distortions.
  This involves distortion or exaggeration of the stimulus or response. This again relates to the heuristic of learning simpler similar tasks first.

- Paired practice of right and wrong answers.
  The incorporation of extra network outputs which are used only during training and have the inverse of the required classification as a target would be equivalent to this. An example would be a second target image with all the non-edges marked, for an edge detecting NNWF.

- Learning a sequence in reverse order of that required in performance.
  The learning of sequences by neural networks involves either a sequence of networks or some form of recurrent network. In the case of a sequence of networks the training order is not important.

Recurrent networks were not investigated during this research but this technique may be helpful in training such networks.

- Learning non-criterion behaviours.
  For example teaching people about electronics when the required behaviour is to repair faulty television sets. This is the education vs. training distinction. Or the distinction between "nice to know" and "required". For ANNs "nice to know" is difficult to define.

Frequency

Frequency of practice examples is clearly of major significance. To make learning "robust" it is generally advantageous not to learn after one example, it may just have been a coincidence. In training ANNs this may also have advantages even though it will increase training times. If a network learns after a single example then any errors in the data set will cause significant difficulties.

6.3.2.3 Treatment Organisation

The ideal treatment organisation provides the minimum amount of practice to produce the required learning. The ideal route being brief practice of the full criterion behaviour, however this is often not possible in which case a sequence of practice tasks will be required. The design of good enabling learning experiences then revolves around the use of three organisational components;

- The individual practice task:
  The individual practice task can use all of the treatment components discussed above. It should also avoid the practising of errors and allow successful completion of aspects of the criterion behaviour.

- Transitions between practice tasks:
  The practice tasks must be progressively more difficult. This clearly relates to the learn simpler similar tasks first heuristic.

- A cumulative series of practice tasks:
  The frequency and variety of the complete set of practice tasks is considered when a cumulative series of practice tasks is designed. The set must culminate in the performance of the criterion behaviour, in the correct mode with the absence of cues and on new examples.

6.3.2.4 Principles

The ID principles should match the treatments and conditions to provide the most effective learning. In the description of conditions and treatments many principles of instructional design have been stated or implied. Gropper suggests appropriate treatments should be specified in three categories; Routine treatments, shaping progressions and specialised treatments. For many learning situations the routine treatments will suffice, in more difficult situations shaping progressions will be required and where even this is not adequate the specialised treatments will be used. The general guidelines for the three categories are given below;

1/ Routine treatments

- cues
- examples
- rules
- practice whole task

2/ Shaping progressions

- unit size reduction
- cues strength varied
- recognise-edit-produce (REP) graduation
- difficulty of examples varied
- practice of easier versions then the whole task

3/ Specialised treatments

- concrete to abstract progressions
- distortions of S and R
- practice errors
- backward chaining
- practice stating principles
- practice of easier versions and then the whole task

The point is made that if the criterion behaviour can be practised at the start then this is the most efficient way to proceed, however there will always be objectives whose requirements exceed what routine treatments can accommodate. The variables that enter into the selection of the instructional strategies include; stimulus and response characteristics, component skills, conditions of performance and types of objectives.

The main classification of objectives being; recalling facts, defining concepts, giving explanations, following procedural rules and solving problems. For each of these objectives variations of the general set of treatments given above are suggested by Gropper.

The objective most closely related to the tasks to be handled by ANNs is "defining concepts" so this will be considered in more detail. The term defining concepts includes defining, identifying and producing relevant instances of a concept. It is the identifying of instances which is most likely to be used by an ANN.

The **learning requirements** for this objective include;

- *Discriminating* instances from non instances.
- Identifying all encountered instances (*generalisation*)
- A definition of the concept has to be learned.
- The skills have to be *transferred* to instances not encountered during training.

The routine treatments suggested for this type of objective are;

- Providing a definition of the concept.

- Providing examples and non examples of the concept.
- Providing a list of defining attributes or principles governing inclusion or non inclusion
- Practice requiring students to state the definition of the concept and either to identify or produce instances.

Non routine treatments will be required if the particular subject matter involved has any of the following characteristics;

- a high degree of similarity between instances and non instances
- a high degree of dissimilarity between instances
- a large number of relevant stimulus properties defining inclusion
- a large number of classes to be discriminated
- a large concept size (a concept containing many instances)

In relating these situations requiring non-routine treatments to training ANNs the first two would appear to relate to the precision with which the inputs must be presented. The next two relate to the dimensions of the input and output vectors respectively. It is difficult to see the relevance of the last characteristic, concept size, except as it impacts the other characteristics. The concept "two cent piece" as opposed to "five cent piece" is unlikely to be more difficult to learn just because there are more two cent pieces in existence, however, the production of a larger number of non standardised objects may have produced a greater variability in physical characteristics, and hence a more difficult concept to learn.

The following items are suggested by Gropper as non routine treatments to facilitate the learning of concepts;

- The use of special cues that help to distinguish instances and non instances,
- recognition practice that pairs instances and non-instances,
- presenting examples in a sequence of graduated difficulty,

- presenting widely differing examples and gradually reducing the difference
- the learning of principles governing the grounds for inclusion or non-inclusion.

All of these relate to the heuristics already identified as useful in training neural networks. The use of principles can, as explained in chapter five, be viewed as a method of providing off-line practice by; generating further examples, defining a simpler similar task or just providing feedback on present performance of the partially trained network or person.

6.3.3 Component Display Theory (Merrill)

Component Display Theory(CDT), as proposed by David Merrill [Merrill 1983], is intended to provide a set of prescriptive relationships which can guide the design and development of learning activities. The theory is at present limited to cognitive subject matter at the micro level. Like the theories of Gagné and Briggs it is based on the idea that different categories of outcomes require different conditions of learning. CDT defines these categories of outcomes as objectives using a two dimensional classification system. The different conditions of learning are provided by a set of appropriate primary and secondary **presentation forms**. Merrill uses the term presentation forms to refer to methods of instruction. CDT is a complex and detailed language that describes the conditions methods and outcomes of instruction. It is also a set of prescriptions which try to optimise achievement of the desired outcomes by prescribing the optimum methods for given conditions.

	Fact	**Concept**	**Procedure**	**Principle**
Find	**X**			
Use	**X**			
Remember				

Figure 6.1: Merrill's Performance - Content Matrix

In Figure 6.1 the Xs represent impossible combinations according to Merrill. The shaded sections are the categories most relevant to training ANNs.

The outcomes are classified using ten categories in what is referred to as a two dimensional performance-content matrix. The performance-content matrix is shown above in Figure 6.1. The prescriptions of the theory however are made on two separate performance and content categorisations and with a modified set of categories in terms of performance. Two questions should be asked in relation to this performance content matrix; Firstly are the two axes actually dimensions with implication of a continuum or at least an ordered sequence or are they just a set of related categories? Secondly are the categories formed by the matrix those most suitable for the purposes of instructional design? That is do they form groups of learning objectives which have different requirements in terms of methods? And are they a good and complete categorisation for this purpose?

In answering these questions the performance and content "dimensions" should be considered separately.

The performance taxonomy suggested by Merrill is based on assumptions about the nature of human memory and learning processes. These assumptions now appear a little dated. For example Merrill refers to searching associative memory, a phrase more appropriate to sequential addressable memory than the more modern concepts of parallel associative memory. The nearest equivalent to 'search time' in associative memory is the time to settle or relax into the appropriate state [Hinton & Anderson 1989].

The extent to which information is integrated into existing knowledge as it is stored is also discussed by Merrill. It is suggested that information can only be integrated with existing knowledge using an algorithmic approach. Research on ANNs has now shown this clearly not to be the case [Feldman 1989].

There is also a difference between the way the categories such as USE are defined and the way they are used in examples of CDT. For example USE and FIND are defined as implying an abstraction, which is why they are not valid for the FACT content category, however if someone has been taught how to clean a clarinet actually doing so does **not** represent any abstraction of the procedure yet it is referred to as an example of **use.procedure**.

Thus while the performance taxonomy may involve some useful groupings it does not appear to form a dimension or be based on a completely valid set of

assumptions. The question arises as to whether other types of performance are relevant to the design of instruction. Other relevant performance variables could be;

- How quickly the learning should take place.
- How long it is to be retained.
- How adaptable to future modification it should be.
- How integrated with existing knowledge it should be.
- How conscious it should be.

The Content or subject matter taxonomy of CDT is based on ideas about the characteristics of subject matter that may be relevant to instructional design. It is suggested that categories or concepts are a form of organisation that is imposed on the world by humans. The types of relationships formed between these concepts is suggested as the basis for the taxonomy of content most suitable for instructional design. The three types of relationships between concepts and the resulting content categories are shown in the table below;

Relationship Type	Description	Content category
identity operations	equality	fact
descriptive operations	subset, intersection	concept
productive operations	ordering in space or time causal (including correlational)	procedure (sequence) principle (rule)

Figure 6.2: CDT Content categories

These content categories appear to have a much more solid basis although again they are a useful set of categories rather than a dimension with any implication of ordering. Again the question of completeness arises. Do these categories encompass all cognitive subject matter?

In terms of the (performance level - content) matrix discussed by Merrill the categories of relevance to the training of the NNWF are **find** and **use** of **concept** or **procedure**. The levels below **use** are analogous to memory systems, either

neural network based or standard address based memories. The **principle** type of content would be appropriate for autonomous systems which had ANNs of the level available today as minor sub units.

6.3.3.1 CDT Methods

Methods in CDT are described in terms of primary and secondary presentation forms or displays. He suggests the two major relevant dimensions for displays are general vs. particular and expository vs. inquisitory. The four categories formed are labeled as shown in the diagram below. Thus material may be presented either in an explanatory(expository) form or as questions in an inquisitory form. Both of these styles can use either generalities or examples.

	Expository(E)	Inquisitory(I)
Generality (G)	EG	IG
Instance or example(eg)	Eeg	Ieg

Figure 6.3: Primary Presentation Forms (PPFs)

The theory suggests that to each primary presentation form a number of elaborative secondary presentations forms can be added. These are shown in the table below.

Secondary Presentation	Description
Context(c)	contextual or background information
Prerequisite(p)	definitions of parts or sub-concepts
Mnemonic(m)	memory aids
Mathemagenic help(h)	help, attention focusing (information. supplied to assist learning)
Representation(r)	alternative representations
Feedback(FB)	answers(ca) mathemagenic(h) use(u)

Figure 6.4: Secondary Presentation Forms (SPFs)

Subsets of this list of secondary presentation forms are suggested as being appropriate for each of the primary presentation forms and are labeled as shown in the diagram below. A ' is added to the mnemonic for PPFs to indicate a secondary presentation form.

	EG	**Eeg**	**Ieg**	**IG**
	Expository Generality	Expository instance	Inquisitory Instance	Inquisitory Generality
Context (**c**)	EG'c	Eeg'c	Ieg'c	IG'c
Prerequisite (**p**)	EG'p	Eeg'p		
Mnemonic (**m**)	EG'm	Eeg'm		
Mathemagenic help (**h**)	EG'h	Eeh'h	Ieg'h	IG'h
Alternative representation(**r**)	EG'r	Eeg'r	Ieg'r	IG'r
Feedback;			FB'ca	FB'ca
correct ans.(**ca**)			FB'h	FB'h
help (**h**)			FB'u	FB'u
use (**u**)				

Figure 6.5: Secondary presentations

6.3.3.2 CDT Principles

The principles in CDT take the form of recommendations as to which primary and secondary display forms are most suitable for a particular performance-content combination. First primary presentation forms are selected based on the required PERFORMANCE category then these presentation forms are qualified or further defined based on the CONTENT category. The parts of these principles specifying primary presentation forms for the FIND and USE levels are shown in the table below.

	Presentation	Practice	Performance
FIND		Iegs.N, IG.N	Iegs.N, IG.N
USE	EG, Eegs	Iegs.N	Iegs.N

Figure 6.6: PPF principles for FIND and USE

In Figure 6.6 the **s** suffix implies two or more examples. The **.N** implies a new previously unseen example or generality

Thus for the USE level of performance the teacher should first present an expository generality then multiple expository instances. In practice and testing multiple inquisitory instances which have not previously been encountered should be used.

For the FIND level of performance there is no presentation but practice and testing should consist of new inquisitory instances followed by inquisitory generalities.

The secondary presentation forms appropriate for each primary presentation form and for the FIND or USE performance level are shown in the Figure 6.7 below. This table can also be stated in words as;

- For the FIND level of performance the only SPF is when asking for a generality; Feedback of form "try to use the generality, see if it works on an instance" should be used.
- For the USE level of performance;

  Attention focusing help, prerequisite information and alternative representations should be used for expository generalities.

  Attention focusing and alternative representations should be used for expository instances.

  Feedback help and alternative representations should be used for inquisitory instances.

	EG	Eeg	Ieg	IG
FIND				FB'u
USE	EG'h, EG'p, EG'r	Eeg'h, Eeg'r	Ieg'r, FB'h	

Figure 6.7: Secondary presentation form principles

A number of other principles, not covered by the tables above, have been added to CDT by Merrill, the key ones being;

Isolation — Tell students which are generalities and which are examples.

Divergence — Make instances as varied as the real population.

Matching — Provide example - non example pairs.
(All irrelevant characteristics of the non-example should match the equivalent characteristics of the example.)

6.3.3.3 Discussion of CDT

Probably the major contribution of CDT is to provide a very detailed shorthand for describing sequences and combinations of strategy components. For example **EG, Ieg, EG'm** would represent an explanatory generality followed by an inquisitory example followed by a secondary presentation providing a mnemonic for an expository generality, however the nomenclature is quite complex and the time investment required to effectively use it is very high.

While it is clear this would be useful for producing or describing computer aided instruction and particularly systems incorporating learner control there would be a considerable initial cost for the learner in becoming familiar with the nomenclature.

Standard training techniques for ANNs provide a sequence of instances that are classified by the network. The result of each classification by the network is compared with the correct classification provided by the training system. Thus two of the primary presentation forms (Eeg & Ieg) are combined. The inability to ask a network for a generality is certainly one of the disadvantages of the NN approach, although systems which try to extract rules from trained NNs overcome this to some extent [Mitra & Pal 1995]. The final PPF of presenting an

expository generality is related to the "using rules" heuristic, which is discussed in the previous chapters [§5.2.3].

The secondary presentation forms of CDT essentially present the strategy components of the other ID theories in another form. Prerequisite information and feedback are clearly important and are covered by the training heuristics suggested. Mnemonic and Context assistance and alternative representations are difficult to apply to training ANNs. They may represent features which are particular to high level systems which make use of language or particular types of symbol manipulation. Divergence is related to the structured training heuristic. The use of matched examples and non-examples could provide a useful extension to the structured training heuristic. The key point here may be ensuring that the matched examples match in all the irrelevant characteristics, this will assist the network to distinguish between relevant and irrelevant input variables or characteristics.

CDT is more difficult to relate to the training of ANNs than the other ID theories reviewed. This may be due to the fact that CDT as described here only claims to cover cognitive subject matter. As discussed earlier, learning of present ANNs is more closely related to the acquisition of motor skills by biological systems.

6.3.4 Elaboration Theory (Reigeluth & Stein)

The final theory of instructional design examined was the Reigeluth - Merrill Elaboration Theory(ET). It is concerned with the macro level of instructional design although the basic principle can also be applied at a lower level. It is concerned with the selection, sequencing, synthesising and summarising of subject matter. It attempts to provide optimal methods in each of these areas. In a similar way to CDT it is intended to provide scope for learner control of selection and sequencing of subject matter and of the timing of synthesis and review activities.

David Ausubel pioneered some important work on the kinds of instructional sequences which help instructional content to be more meaningful to the learner and hence easier to learn [Ausubel 1968]. He advocated initiating instruction with general level knowledge that subsumes the content that is to follow; the remainder of the instruction is a process of successive differentiation.

Elaboration Theory presently uses seven major strategy components:

1. A special type of simple to complex sequence for the main structure of a course.
2. Learning-prerequisite sequences within individual lessons of the course.
3. Summarisers.
4. Synthesisers.
5. Analogies.
6. Cognitive-strategy activators.
7. A learner control format.

Just the first two aspects of ET are considered in detail as the remainder are not relevant to the training of ANNs at this stage of the development of the field.

An elaborative sequence is a special kind of simple to complex sequence in which;

- The general ideas epitomize rather than summarise the ideas that follow.
- The epitomizing is done on the basis of a single type of content.

Epitomizing entails presenting a very small number of the ideas that are to be taught, at a concrete meaningful application level. Thus epitomizing is to teach a few fundamental and representative ideas at the application level. The application level is the **use a generality** level. Epitomizing is done with just one of the three types of content: concepts, procedures or principles. The term epitome is used to refer to this very special kind of overview which involves identifying ideas which are either very simple or very general but never abstract.

In the case of teaching concepts the epitomizing approach is one in which only the most general and inclusive concepts are taught in the overview at the application level. This is followed by elaborating out to the remaining concepts at a less inclusive and more detailed level.

In the case of teaching procedures an epitomizing approach entails presenting the shortest path first at the application level and then elaborating it out to the desired breadth and complexity of alternative paths.

For principles the epitomizing approach is one in which the most simple and fundamental principles are taught first, at the application level. This is followed by elaborating out to the remaining principles. This turns out to often produce a similar order to that in which the principles were discovered. Part of the rational for the epitomizing approach is that it provides for creation of meaningful contexts within which subsequent instruction content can be acquired. It is this aspect which seems most relevant to the training of ANNs. In this way it appears to be an extension of the structured training heuristic in which the sequence of training examples is chosen such that it forms an epitomizing sequence of instruction.

ET also incorporates learning-prerequisite sequences which almost seem to conflict with the epitomizing approach. ET defines learning-prerequisites as the critical components of an idea. The critical components of principles are;

- concepts
- change relationships

The critical components of concepts are;

- defining attributes
- their interrelationships

The critical components of procedures are;

- more detailed descriptions of the actions in a step
- concepts that relate to the actions
- more detailed descriptions on the factors influencing decision steps
- concepts that relate to those factors
- rules for considering the factors in making a decision.

A learning-prerequisite sequence is a sequence in which an idea is not presented until all of its learning prerequisites have been presented. In this way the ET reinforces the idea that the learning of prerequisites should be confined to truly **critical** components that **must** be learned first. It also identifies the type of critical components involved for each of the content types.

This idea can be related to the heuristic for learning useful sub-tasks. The term critical component is better than learning prerequisite as while important these components do not **have** to be learnt first. For example the network learning a concept must certainly be able to discriminate in terms of the defining attributes. It must also be capable of forming the interrelationships required. Thus a single layer network that cannot form an exclusive OR cannot learn a concept which is defined by an exclusive OR of two attributes. It does not, however, have to learn either the OR function or the attributes separately before learning the whole task. Thus it may be more important to determine whether the critical components can be learnt by the proposed network rather than to actually teach the network these components.

The question of when it is useful to either learn a sub-task first or train a sub network to perform the sub-task still requires further investigation. Some experiments and further discussion of this point are contained in the following chapters.

6.4 Discussion of ID theories

The previous section briefly describes a variety of theories of instructional design. The question to be asked is what are the common elements of these theories in terms of ID for ANNs? How do they relate to the proposed heuristics? And what additional suggestions for training ANNs do they contain.

Of the four ID theories discussed the behavioural approach is most easily related to ID for ANNs, however, they all appear to have common elements and contain the four heuristics in one form or another. The behavioural approach is perhaps most appropriate because it views training from a relatively low level. This is certainly appropriate for today's ANNs that would be struggling to compete with a cockroach in terms of learning ability.

The key features that appear in common among several of the theories examined are;

- Learning prerequisites or useful sub-tasks.
- Practice with feedback.
- Standard treatments for tasks which turn out to be relatively simple to learn and various specialised treatments for use in more difficult situations.

The important features which only appear in individual theories examined are;

- Cues
- Elaboration sequences.

Based on these points an extended set of heuristics or strategy components is presented in the next chapter.

6.5 A THEORY OF ID FOR ANNS

To build a theory of instructional design for artificial neural networks the general process of building an instructional design theory should be examined first. Then the modifications required when training ANNs rather than people should be considered. The particular task domain of this thesis, window filters to be used in image processing, may also affect the process.

Reigeluth suggests four steps are involved in the construction of a theory of instructional design these are;

1. Develop formative hypothesis.
   data, experience, intuition, logic

2. Develop a taxonomy
   Name possibly relevant variables and a provisional system of classification.

3. Derive principles
   experience, intuition, logic -> postulating principles
   empirical research -> test principles

4. Develop models and theories
   Integrate principles from step three.

The work discussed in this thesis progresses part way into stage three. Thus experience, in the form of a series of experiments aimed at training a NNWF to perform various tasks, together with some application of intuition and logic is just starting to enable us to postulate some possible principles. Further empirical research will enable us to test these principles. At some later stage it may be possible to integrate the various principles and modify or enhance the classification schemes for both situations and methods to provide a coherent theory of instructional design for artificial neural networks. To some extent it is quite reasonable to expect this to be an iterative process. As we look for possible principles and try to combine them this may well suggest refinements to the methods of classification which will in turn allow new principles to be formed.

As a practical approach to formulating the instructional design to enable a particular task to be performed by an artificial neural network the following structure is suggested.

1. **Perform a task analysis to determine the likely difficulty of learning the task.** In this step consideration of aspects suggested by Gropper should be helpful; Generalisation, discrimination, stimulus response characteristics, etc. From this it will be possible to make an initial estimate of whether a standard approach will suffice or whether some form of special treatment is likely to be required.

2. **Consider the stimulus-response characteristics.** This includes consideration of the use of fuzzy targets and the choice of a suitable coding form for inputs and outputs. Coding should be chosen to give sufficient resolution, to make input patterns that are to be discriminated as different as possible, and input patterns that are to be generalised as similar as possible. The use of self organising networks to pre process the input data should also be considered.

3. **Initial trial training runs** should then be run.

4. **If training is not successful additional strategy components should then be used.** These can be planned under the general headings of the four basic heuristics suggested in the next chapter, which are extended versions of those already discussed.

5. **The results from training runs using the extended heuristics should then be critically examined.** The forms of failures need to be examined and used to modify the training environment. This is similar to the approach suggested by Gropper [Gropper 1975 Ch 4] for evaluating and improving human training presentations.

Consideration will also need to be given to the neuron model, network structure and learning algorithm to determine whether these are limiting performance, however this is not the focus of this work.

The identification of these strategy components or heuristics and the variety of methods by which they can be implemented as part of the NNWF is the easiest place to continue the development of an instructional design theory for ANNs. The following chapters look at the definition of key strategy components, some

experiments that attempt to demonstrate the use of the different strategy components and the implementation methods available.

6.6 REFERENCES

page first cited

Aronson & Briggs 1983
D.T. Aronson, L.J. Briggs, *Contributions of Gagné and Briggs*, in Instructional-Design Theories and Models, Lawrence Erlbaum Associates, New Jersey, 1983, pp75-100 ..... 6.9

Ausubel 1968
D.P. Ausubel, Educational Psychology 6.32

Briggs 1977
L.J. Briggs, Instructional Design 6.9

Dorner 1983
D. Dorner, Heuristics and cognition in complex systems, in Methods of heuristics, Lawrence Erlbaum Associates, New Jersey, 1983 ..... 6.14

Entwistle & Ramsden 1983
N.J. Entwistle, P. Ramsden, Understanding Student Learning, Croom Helm, Beckenham Kent, 1983 ..... 6.6

Fahlman & Lebierer 1990
S.E. Fahlman, C. Lebierer, *The cascade correlation learning architecture*, in Advances in Neural Information Processing Systems 2, Morgan Kaufmann, Los Altos CA, 1990, pp524-532 ..... 6.6

Feldman 1989
J.A. Feldman, *A Connectionist Model of Visual Memory*, in Parallel Models of Associative Memory, Lawrence Erlbaum, New Jersey, 1989, Ch 2 ..... 6.25

Gagné & Briggs 1974
Robert M. Gagné, Leslie J. Briggs, Principles of Instructional Design, Holt Rinehart & Winston, New York, 1974 ..... 6.9

Gropper 1975
G.L. Gropper, Diagnosis and Revision in the Development of Instructional Materials, Educational Technology Publications, New Jersey, 1975 ..... 6.38

Gropper 1983
G.L. Gropper, *A Behavioural Approach to Instructional Prescription*, in Instructional-Design Theories and Models, Lawrence Erlbaum Associates, New Jersey, 1983, pp101-161 ..... 6.13

Hinton & Anderson 1989
G. Hinton, J. Anderson, Parallel Models of Associative Memory Updated Edition, Lawrence Erlbaum, New Jersey, 1989 ..... 6.25

Hrycej 1992
T. Hrycej, *Supporting supervised learning by self-organisation*, Neurocomputing, Vol. 4, No.1-2, 1992, pp17-30 ..... 6.16

Kohonen 1990
T. Kohonen, *The Self-Organizing Map*, Proceedings of the IEEE, Vol. 78 No.9, Sept 1990, pp1464-1480 ..... 6.16

Merrill 1983
M.D. Merrill, *Component Display Theory*, in Instructional Design Theories and Models, Lawrence Erlbaum Associates, New Jersey, 1983, pp279-333 ..... 6.24

Mitra & Pal 1995

S. Mitra , S.K. Pal, *Fuzzy Multi-Layer Perceptron, Inferencing and Rule Generation*, IEEE Transactions on Neural Networks, Vol. 6 No. 1, Jan 1995, pp51-63 ..... 6.31

Møller 1993

M.F. Møller, *A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning*, Neural Networks, Vol. 6 No. 4, 1993, pp525-533..... 6.6

Ngan 1993

P.M. Ngan, The Development of a Visual Language for Image Processing Applications, Thesis, PhD, Massey University, 1993 ..... 6.14

Ooyen & Nienhuis 1992

A. Van Ooyen, B. Nienhuis, *Improving the Convergence of the Back-Propagation Algorithm*, Neural Networks, Vol. 5, No. 3, 1992, pp459-463..... 6.6

Ratcliff 1990

R. Ratcliff, *Connectionist Models of Recognition Memory: Constraints Imposed by Learning and Forgetting Functions*, Psychological Review, Vol. 97 No. 2, 1990, pp285-308..... 6.16

Reigeluth 1983a

C.M. Reigeluth, *Instructional Design: What is it and why is it?*, In Instructional-Design Theories and Models, Lawrence Erlbaum Associates, New Jersey, 1983, pp3-36..... 6.2

Riedmiller & Braun 1993

M. Riedmiller, H. Braun, *RPROP: A Fast and Robust Backpropagation Learning Strategy*, in Proceedings of the fourth Australian Conference on Neural Networks ACNN'93, Melbourne, Jan 1993, pp169-172..... 6.6

Robins 1993

A. Robins, *Catastrophic Forgetting in Neural Networks: The Role of Rehearsal Mechanism*, in Proceedings of Artificial Neural Networks and Expert Systems ANNES'93, Dunedin, 1993, pp65-68 ..... 6.16

Sutton 1992

Richard S. Sutton, Reinforcement Learning, Kluwer Academic Publ., Boston, 1992 ..... 6.13

Tollenaere 1990

T. Tollenaere, *SuperSAB: Fast adaptive back propagation with good scaling properties.*, Neural Networks, Vol. 35, 1990, pp561-573..... 6.6

Whitehead & Ballard 1990

S.D. Whitehead, D.H. Ballard, *Active perception and reinforcement learning*, Neural-Computation, Vol. 24, 1990, pp409-419..... 6.13

Chapter 7

7. Training Heuristics and their Implementation

The analysis in the previous chapter shows that the four heuristics suggested earlier have a firm basis in instructional design theory, at least as it applies to people. In this chapter these heuristics are restated in an expanded form. In addition, the idea of using fuzzy targets is developed into a full heuristic which emphasizes the importance of stimulus response characteristics. In this new form the five heuristics proposed incorporate the key ideas from instructional design theory and form a logical grouping of strategy components for improving the training of artificial neural networks.

7.1 INTRODUCTION

The word **heuristic** is derived form the Greek *heurisko* or find, the Concise Oxford Dictionary provides the definition as; *serving to discover*. Therefore a heuristic is a rule of thumb by the use of which a blind trial and error search is raised to a more directed or intelligent search for a solution.

It is in the sense of *serving to discover* that the word heuristic is used here. By the use of these heuristics the likelihood of the neural network, together with its training environment, discovering a solution to the task should be enhanced. In this vein the following five heuristics, are suggested for use in training artificial neural networks;

1. Consideration of stimulus / response characteristics.
2. Learning of useful sub-tasks first.
3. Learning simpler, similar tasks first.
4. Use of rules to assist learning.
5. The use of structured training.

The first heuristic is not so much a strategy for training as a strategy for controlling the formulation of the task and determining the need for special training techniques. The second two heuristics represent the most straightforward ways of enhancing training; learning useful sub-tasks first and pre-training on simpler similar tasks. The fourth item covers the use of rules for; generating additional examples, generating simpler similar tasks, specifying useful sub-tasks and for incorporating rule based knowledge directly into the training. The last item; the use of structured training, covers the incorporation and sequencing of the other techniques or of specialised sets of training examples.

In this chapter each heuristic is explained and implementation methods developed. The description of both the heuristics and their implementation places conflicting requirements on the layout of the chapter. The heuristics are described in the order in which they should be applied, however, in terms of implementation they form two logical groups.

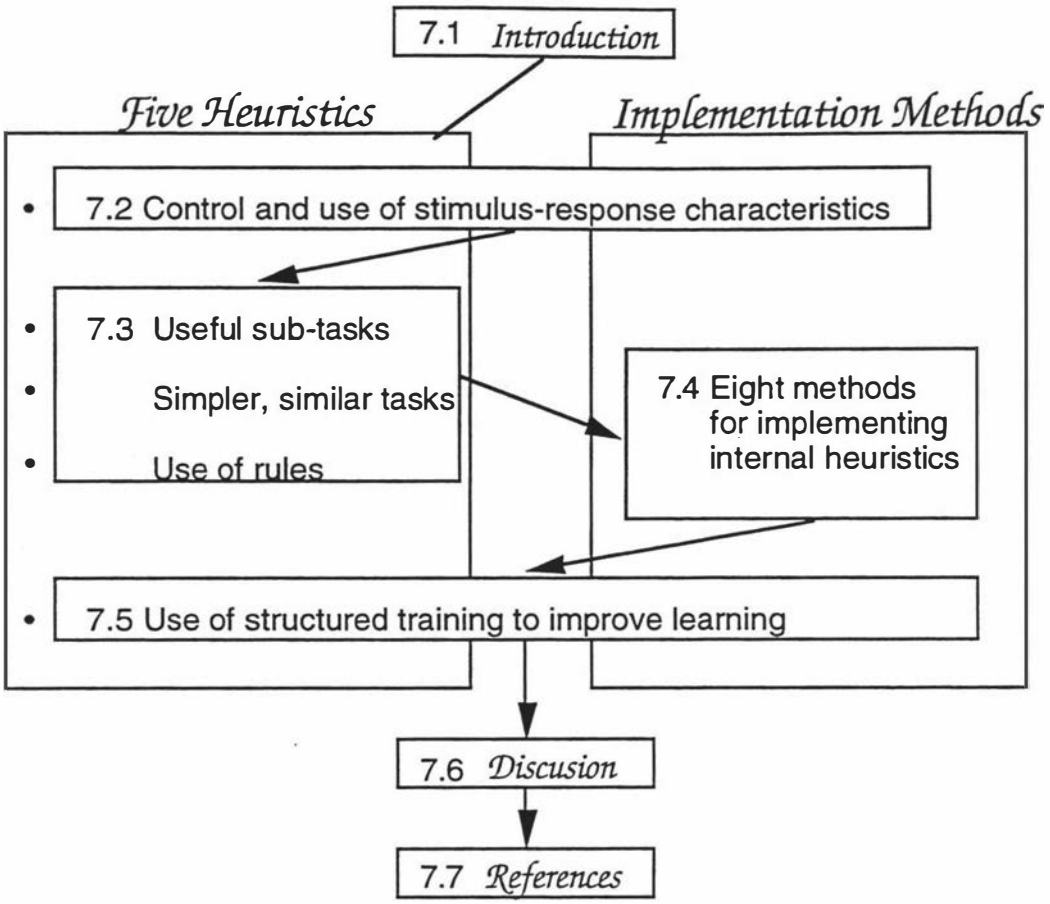
The term "network" is taken to include the following items;

- a neuron model
- a topology of connectivity
- a learning rule
- a training environment

The heuristics can then be classified into two groups; those which are independent of the network, which can be called **external heuristics**, and those whose implementation will be dependant on the type of network being used, which can be called **internal heuristics.**

By these definitions heuristics two, three and four are internal heuristics. Whereas heuristics one and five are external heuristics which take account of external training factors. The internal heuristics have a number of implementation methods in common. For this reason the three internal heuristics are described together in section 7.3 then the implementation methods for them are described in section 7.4. For the external heuristics implementation is specific and independent of the network type. For this reason the first external heuristic and its implementation is described in section 7.2 The other external heuristic (on structured training) and its implementation is described in section 7.5.

This logical structure of the chapter is shown diagrammatically below.



To experimentally investigate the usefulness of these five heuristics they were incorporated into the neural network window filter and its training environment. A variety of methods for implementing each heuristic were investigated and are discussed. The chapter finishes with a brief discussion of general issues and a list of references. In the following chapter, a set of experiments is described which demonstrate the use and implementation of this extended set of heuristics.

7.2 Consideration of Stimulus / Response Characteristics

It was realised from the discussion in the previous chapter that the use of fuzzy targets is but one example that could arise from a more general consideration of stimulus response characteristics.

In the context of the artificial neural network the stimulus and response of instructional design theory are analogous to specific sets of input and output vectors. The stimulus is that set of input vectors which is to be responded to in a particular way. The response is the required output vector or set of vectors. In a classification task the stimulus may be all the instances or input vectors, both those used in training and those encountered later, that are to be classified in a particular way. The response is the output vector which represents this classification. In another type of task, for example the addition of two numbers, the term response refers to the set of output vectors which represent the sum of the inputs. In this case the actual output value will be different for different input combinations, but the set is still referred to as the required response.

The conflicting requirements of discrimination and generalisation are of major importance here. The S-R characteristics which effect this are those described as conditions by Gropper in the previous chapter [§ 6.3.2].[1] They are restated here in terms more appropriate to the training of ANNs:

> To enable **generalisation** to take place, the input vectors for instances which are to provide **similar** responses should be as similar as possible.
>
> To enable **discrimination** to take place, the input vectors for instances which are to provide **different** responses should be as dissimilar as possible.
>
> The input vector should only contain items which are required to produce the correct response. It should contain as few elements as is compatible with the previous two requirements for ease of discrimination and generalisation.

[1] These requirements are also covered in other ID theories but are described in different terms.

These requirements may lead to the use of fuzzy targets, input coding and pre-processing. Consideration of the S-R characteristics is part of the task analysis stage in instructional design(for people) to determine whether a specialised treatment will be required. A similar approach is possible in instructional design for artificial neural networks. In ID for ANNs consideration of S-R characteristics may allow a choice of which inputs and outputs to use, how to represent them and what coding scheme to use. The extent to which these are under control of the neural network instructional designer will vary, however at least some of these aspects are generally not fixed by the nature of the problem. By carefull selection of which inputs and outputs are used and how they are represented (or coded) the network performance may be improved. A network which would not previously converge may then do so, convergence may be faster or the solution accuracy may be increased. Where certain S-R characteristics cannot be avoided, their presence may suggest the need for specialised training regimes which make use of the heuristics covered in the following sections of this chapter. Finally, it may be possible to pre-process the inputs before presenting them to the network. This is done to improve the S-R characteristics of the task as presented to the network. Self organising networks have been shown to provide useful transformations for this pre-processing stage [Hrycej 1992, Kohonen 1990, Grandvalet et al. 1991] but analytical and algorithmic methods can also be used.

The implementation of each of these techniques is considered in the following paragraphs.

7.2.1.1 Input/Output Coding

The application of an ANN to any specific task involves the choice of input output coding techniques. These choices should be made explicit while keeping the discrimination and generalisation requirements in mind. That is the choice of coding techniques should be taken as a specific step in applying a network to each new task. The choice should be made so as to ensure that

a) the input vectors for instances which are to provide **similar** responses are as similar as possible.

b) the input vectors for instances which are to provide **different** responses are as dissimilar as possible.

This can be achieved by choosing from the following forms of coding;

- Direct - A single number is used for each network input or output, normalised to the range of either 0-1 or -1 to +1 depending on the squashing function used in the neuron model. For example in the NNWF the input and output pixel intensities which range from 0 to 255 are scaled to cover the range 0 to 1.

- Binary - When true or false values are to be used as inputs or outputs the most straightforward approach is to use 0 for false and 1 for true. Often values of 0.1 and 0.9 are substituted to avoid some of the inherent limitations in standard back propagation when nodes are near saturation in either direction [Rumelhart & McCleland 1986].

- Logarithmic compression or scaling - This form of pre-processing allows a large dynamic range to be represented. There is considerable evidence for the use of logarithmically scaled sensory signals in biological systems [Cornsweet 1970]. Logarithmic scaling also affects the way in which inputs interact in the network. For example addition produces, in effect, a multiplication and subtraction provides a ratio of the raw values.

- Binary encoding - This can be used for both inputs and outputs and involves replacing a single connection by a vector each component of which represents one binary bit. Each of these bits is then considered as a separate input or output. Again the binary values may be internally represented by 0.1 and 0.9 rather than 0 and 1. The discrimination and generalisation required should be considered carefully when using this coding technique as a small change in a parameter value may result in a step change in the binary encoded representation. This may aid or hinder learning. It will be helpful when the values of the parameter do not represent some form of continuum. For example consider a parameter which represents the part number of inventory items which are not ordered into groups. In this case the input for two parts which have adjacent part numbers should be made quite different to aid in discrimination.

- Interval coding - Where the effect of a parameter is known to vary in different regions of its range it may be advantageous to use a separate input or output for each region. A number of inputs (or outputs) is

used to represent the entire range and each unit is active for only part of the range.

- Fuzzy Coding - Fuzzy inputs and outputs can often be used to advantage to eliminate the artificially sharp steps between essentially similar inputs or outputs. This can help in generalisation.

  When used for input variables the membership functions are constructed for each of the fuzzy concepts and then these are used to transform a single input value into a number of derived values. For outputs a single target value is decomposed in a similar way to provide targets for each of the fuzzy outputs. When running the network the fuzzy output may be used directly or a maximum chosen or some more complex de-fuzzyfication technique used.

7.2.1.2 Pre-processing network inputs

As mentioned earlier pre-processing can be achieved by using either additional neural networks or algorithmic manipulation of the input variables. Algorithmic pre-processing can be considered as just another form of input coding. Logarithmic coding as described above is an example of this. Other examples are the pre-calculation of powers and cross multiples of input variables. A successful application of this technique is described by Tuck [Tuck 1993].

Another very useful form of pre-processing is the use of self organising networks to extract significant features. This is not considered in detail here but useful examples are provided in the references given above [Hrycej 1992, Kohonen 1990, Grandvalet et al. 1991]

7.3 Internal Heuristics

This section explains the three internal heuristics which lead to modifications of the NNWF and the environment in which it is trained. The implementation of these heuristics is covered in section 7.4.

7.3.1 Learning Useful Sub-Tasks First

A useful sub-task is any task whose execution may assist in the performance of the entire task. For example, finding edges may be a useful sub-task in identifying cancerous cells. A sub-task may be part of a sequence of operations which will as a whole perform the desired function. For example, a simple function such as AND, can be considered a sub-task. If other parts of the network can determine which points are edges and which points are part of malignant cells then an AND sub-task could be useful in determining which parts of the image are the edges of cancerous cells.

The concept of a sub-task as presented in chapter five is extended here in the light of ideas from instructional design. Several theories discussed in chapter six refer to the use of pre-requisite skills. Different words are used to describe similar concepts in the discussion of cues, incrementing and fading in the behavioural theories of instructional design. The term pre-requisite skills is generally not appropriate for ANNs because it implies the existence of a known solution method which requires the use of these prerequisite skills. With an ANN, the more usual situation is that of developing enabling skills or capabilities in anticipation of their possible use in an unknown solution method. The network teacher specifies these potentially useful sub-tasks in the anticipation that their incorporation into the network will aid in learning the target task. If the choice of sub-task was appropriate it will either speed up learning or actually enable the learning of a previously unlearnable task.

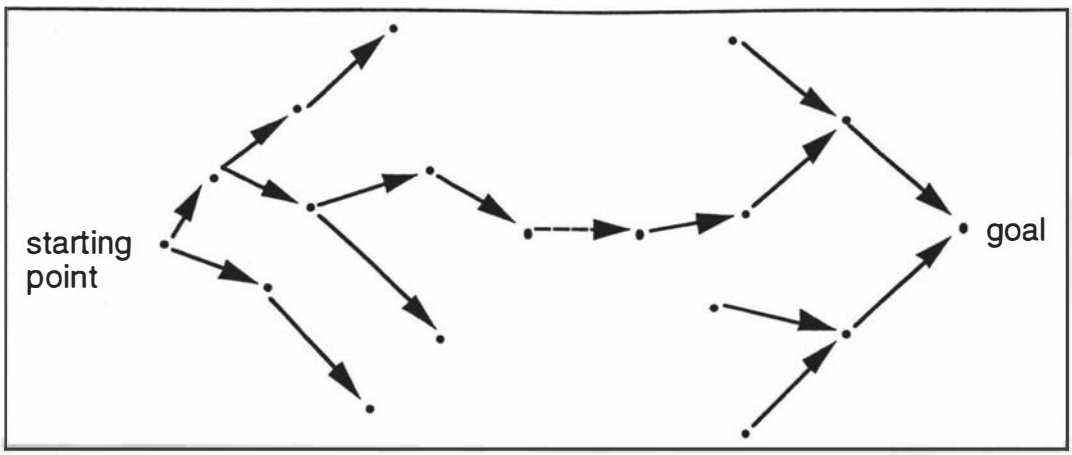


Figure 7.1: Solution Graph

A useful approach is to consider sub-tasks as elements in a solution graph (see Figure 7.1 above and the discussion of incrementing and the use of cues in §7.3.2). The role of this heuristic is then to provide the network with possibly useful links in this graph which will help it form a connection between given inputs and desired outputs. The key methods of doing this effectively is to build inwards from both ends of the solution graph. To use a cue during training is to build from the output end towards the middle of the graph. A cue being the input to a sub-task near the end of the solution graph. Incrementing refers to the building of a series of sub-tasks starting from the input end and moving towards the centre of the graph. The approach of building a solution graph from both ends has been found helpful in image processing algorithm development [Ngan 1993, Dorner 1983] and may therefore be particularly relevant for a neural network for image processing tasks.

The methods by which sub-tasks may be incorporated into the NNWF training using either sub-networks, sub algorithms or a modified network structure are discussed in detail in section 7.4.2.

7.3.2 Learning Simpler, Similar Tasks First

This heuristic concerns training on simpler, similar tasks before attempting to learn the main task. The intention is that the learning of the simpler task will either speed up or actually enable the later learning of the main task. This heuristic incorporates the ideas of shaping, distortions, coarse discrimination, over generalisation and epitomes (as described in chapter 6). Shaping is the learning of a series of approximations, each being closer to completion of the final desired task. Coarse discrimination refers to the process of starting training with the presentation of particular pairs of examples in which the different

classes are clearly differentiated. Over generalisation refers to the initial over generalisation found in most animal learning [Catania 1992, Manning 1979]. Distortions are examples which have been modified to accentuate the differences between classes. Epitomes are classic concrete examples which epitomise the required concept or operation. Epitomes are carefully chosen not to incorporate any of the subtleties of more complex examples. These are all forms of simpler, similar, tasks; some are based on the selection of examples and some are based on the modification or creation of special examples.

A variety of methods for the incorporation of the prior learning of these simpler similar tasks into the training environment for the NNWF have been devised. These are discussed in detail in section 7.4.3

7.3.3 Use of Rules To Assist Learning

The role of rules in human learning and cognition has been discussed in detail in chapter four. It is suggested that the use of rules should be seen predominantly as a method of transferring knowledge and skills from one person to another. For this reason the use of rules forms an essential part of an educational environmen for ANNs. This heuristic concerns the ways in which a teacher can make use of rules to improve the training of ANNs. Five ways of using rules to transfer knowledge have been identified.

1. Using a rule to produce additional examples.

2. Using a rule to define a useful sub-task.

3. Using a rule to define a simpler, similar task.

4. Using a rule to directly constrain the network structure.

5. Using a rule to provide direct input of error without the use of a target.

7.3.3.1 Using rules to produce additional examples

Perhaps the simplest use of a rule is to define further examples which can then be used for training in the normal way. This is valuable when insufficient examples are available or when "real world" examples are difficult or expensive to generate. Training using real world examples may also be dangerous. For example consider learning to classify animals into those that eat people and those that do not. Any way of avoiding the use

of real world examples for at least the early stages of learning this task will have advantages such as staying alive! Similar examples more relevant to training artificial neural networks can easily be envisioned in for example neural network based control systems.

The use of rules in this way does not require that a general algorithm for performing the task should already exist. An example for training an ANN consists of a matched input – target pair. It does not require that the output is generated from the input. Therefore a rule that generates inputs from outputs is equally useful. For example if we want to train a network to find square roots we can use a rule for generating squares to generate examples by picking random output values and generating corresponding inputs as squares. The only requirement for a rule is that in some way it allows identification of suitable training pairs. As the rule is only used for generating examples and not in the final operation, the complexity of the rule is also of less importance. For this reason a very complex rule which requires a time consuming sequence to be generated can also be usefully used to generate training pairs. The generated training pairs may then be used to train a network. By operating in a parallel form the implemented network may then perform the same task but at much higher speed.[2]

7.3.3.2 Using rules to define a useful sub-task

The second method of using a rule is to use it to define a useful sub-task. The sub-task identified may be of any of the types identified in section 7.3.1. For example, a network which is to sort letters at a postal centre may be able to make use of information regarding the orientation of the lines of text. The orientation can be determined by integrating pixel values along a series of parallel lines at a range of orientations and picking the direction which provides maximum contrast. The sub-task identified in this way may then be incorporated into the NNWF in any of the ways described in section 7.4.2 for implementing general sub-tasks.

---

[2] This may also allow us to use a biological brain, which can only handle logical sequential thought in a slow and laborious way, to perform tasks which can easily be described by rules.

7.3.3.3 Using rules to define a simpler, similar task

Rules can be used as a method of transferring knowledge from one person to another by describing the behaviour of a functioning network. This description may incorporate a simplification or approximation. The third way of using rules is thus to define a simpler similar task. This is perhaps the main way in which rules are used in human learning. When Mary tells her son to "put your foot on the clutch before moving the gear lever, and then release the clutch smoothly" she is not suggesting that this is all that is required to produce a smooth gear change. What she is suggesting is that this is a rough approximation to the behaviour required and that until you have that embedded in your mind it would be better not to try your neural network on the real thing, for example her expensive new car!

7.3.3.4 Using rules to directly constrain the network structure

The fourth use of rules is to identify constraints on the structure of the network. This will in general provide a network with fewer degrees of freedom. Such networks, it is suggested, will be easier to train. For example a rule may imply that a window filter should respond in the same way to items which are reflected in a vertical plane. In this case the weights from opposite sides of the input window should always be the same. This constraint can be incorporated into the structure of the network. For example the pairs of weights involved might be replaced by a single weight which is used for both connections in the network, alternatively the individual weights might be combined in some way at intervals during the training.

Le Cun describes an example of the use of rules in this form [Le Cun et al. 1989].

7.3.3.5 Using rules to provide direct indication of error

Rules or hints will sometimes allow the direct production of an error signal without the generation of a target. The back-propagation rule used for training neural networks only requires an output error for calculation of weight changes. This is normally generated from a difference between the target and the actual output values, however there is no requirement in the derivation of back-propagation that the error signal be generated in

this way. Abu Mostafa has described the use of rules or hints in this manner [Abu-Mostafa 1993].

The detailed methods by which these various uses of rules can be incorporated into the NNWF training environment are discussed in section 7.4.4.

7.4 Implementation Methods for Internal Heuristics

The methods of implementing heuristics two three and four are discussed here. The eight methods identified are not all applicable to all three heuristics. Figure 7.2 identifies which methods are applicable to which heuristics. To provide a framework for the later discussion the eight methods are first described in section 7.4.1. The use and further properties of each of the three internal heuristics are considered in sub sections 2, 3 and 4.

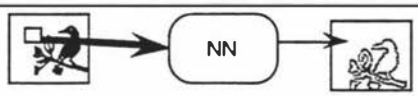
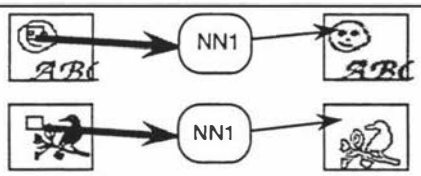
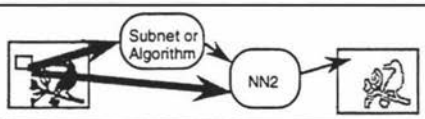
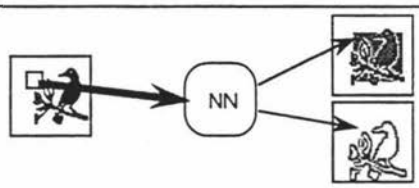
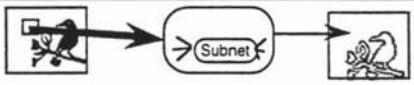
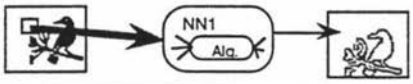
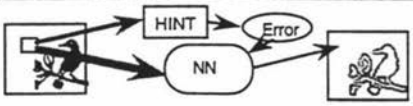
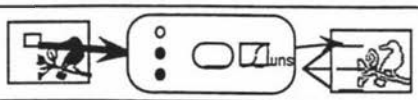
Heuristics -> Methods	Diagram	H0 Examples	H2 Sub-tasks	H3 Simpler Similar	H4 Rules
M0 Examples		Simple throw examples at it strategy			
M1 Sequence			Can only be used if the final output can be derived from just sub-task output	Can only be used if similar task is also a sub-task and if the final output can be derived from just sub-task output	Rule defines pre processing from which output comes
M2 Pretraining			Will only help if sub-task is also a simpler similar task	Main method of applying simpler similar. Could be a sequence of progressively harder tasks	Using rules to generate simpler similar task
M3 Dual Inputs			(1) Three main methods of applying sub-tasks	Just using simpler similar task as a sub-task	(2) Four ways of using rule to define sub - task
M4 Dual Outputs			Sub-task may form hidden units into useful functions	Sub-task may form hidden units into useful functions	(2) Four ways of using rule to define sub-task
M5 Embedded Sub Network			(1) Three main methods of applying sub-tasks	Just using simpler similar task as a sub-task	(2) Four ways of using rule to define sub-task
M6 Embedded Sub Algorithm			(1) Three main methods of applying sub-tasks	Just using simpler similar task as a sub-task	(2) Four ways of using rule to define sub task
M7 Direct Use of Hints					Using the HINT method of rules to provide direct error
M8 Structural Changes					Using rule to constrain net structure

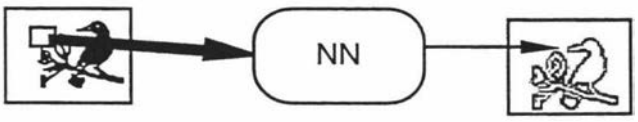
Figure 7.2: Implementation of Internal Heuristics

7.4.1 Eight Methods for Implementing Internal Heuristics

Eight implementation methods have been identified for the three internal heuristics that is sub-tasks, simpler similar tasks, and the use of rules. The methods are identified as M1 to M8 and are listed below each with a brief title for identification. Each method is then described in detail. For simplicity each method is initially described with reference to learning a sub-task and where this is not appropriate, with reference to its most common use. The standard method of training by providing a series of examples(M0) is also described here to provide a point of reference for description of the new methods.

- M0 Standard Training Examples
- M1 Sequence of Networks
- M2 Pre-Training
- M3 Dual Inputs
- M4 Dual Outputs
- M5 Embedded Sub-Network
- M6 Embedded Sub Algorithm
- M7 Direct Use of Error or Hints
- M8 Structural Changes to Network

7.4.1.1 M0 Standard Training by Examples



As described in chapter three the NNWF is trained by presentation of an input and a target image. As the window is scanned across the input image, the set of pixels from the window at each position in the input image provides an input vector to the network. The corresponding target pixel provides a target output value. This process is shown diagrammatically above. Presentation of a single image of dimensions x and y with a window size of w provides (x-w+1)*(y-w+1) training pairs or examples. These are presented to the network in an order determined by the normal raster scanning sequence. Each scan through the

entire image is referred to as an epoch. It should be noted that this sequencing of the training examples is not random and will sometimes help and sometimes hinder the learning of the required mapping.

7.4.1.2 M1 Sequence of Networks

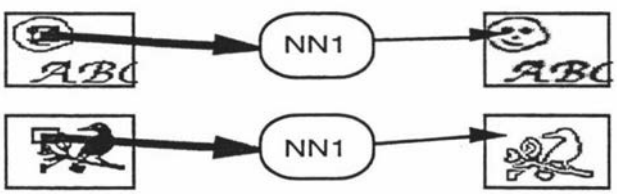
The first new method is simply to use a sequence of two networks to allow the incorporation of a sub-task. The first network learns the sub-task and the output from this network is used for the input to a second network which is to provide the main output. If a series of steps are required then the two networks are replaced by a sequence of networks. Each network receives inputs from the output of the previous one. Such a system is shown diagrammatically above as it applies to the NNWF.

This approach clearly has limitations;

- The sub-task or tasks must form part of a sequence.
- The output of the sub-task must provide all the information required to perform the following tasks and hence produce the final output.

While this limits this approach to special kinds of sub-task there are situations in which this method is adequate and an example is discussed in the next chapter [§8.2.1]. Where the following tasks will require additional information from the original image one of the more complicated methods such as M3, M5 or M6 will have to be used.

7.4.1.3 M2 Pre-Training



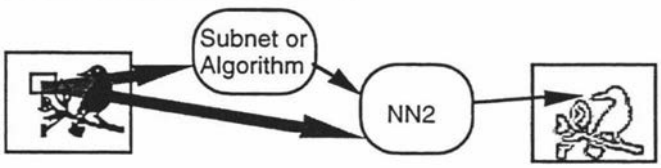
The next very simple modification to the normal method of training is to pre-train on a different task, or on a different set of examples. A network may be trained to perform a particular sub-task. Once this sub-task is at least partially

learnt, this network with learnt weights can be used as the starting point for training on the whole task. For example a network might be trained to find all edges in images. This pre-trained network would then be used as a starting point for learning to find the edges of apples in images of apple trees.

The error surface for the main task is not modified by pre-training. The effect of pre-training is to start the network in a different position in error space. This may have two non-exclusive benefits; It may help to avoid the network becoming trapped in local minima. It may provide faster convergence due to the network starting in a position which is well within a basin of attraction of the optimal solution.

This method is most appropriate for an implementation of some of the other heuristics such as simpler similar tasks. It is perhaps surprising that it works at all for general sub-tasks as the same network is being trained to perform one task and then training is switched to make it perform a different task. The success of this method will clearly depend on the nature of the sub-task. Even when they appear quite dissimilar on the surface the pre-training task may bias the network to start the search for a solution in a more propitious region of weight space. Examples of the use of this method are described in chapter eight.

7.4.1.4 M3 Dual Inputs

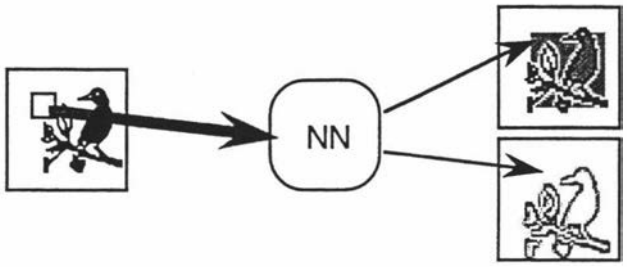


A dual input network is simply a network in which a second set of inputs is provided. When learning a sub-task the additional inputs allow the sub-task to be handled either by an additional sub-network or by an algorithm. The outputs of the unit performing the sub-task are connected as additional inputs to the main network. A NNWF with an algorithmic or subnet sub-task is shown diagrammatically above.

This method of implementation is similar to that used when coding the inputs to a network but provides both the original and the coded inputs. This can be helpful if it is only one aspect of the information available in the inputs which is being coded by the sub-task. For example, a particular task may have two inputs x and y. If the main task involves the product of x and y but also uses the two

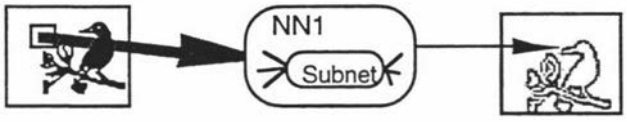
terms individually then a dual input network which accepts an xy term in addition to the original x and y inputs may be appropriate. Examples of the use of this method are given in the following chapter.

7.4.1.5 M4 Dual Outputs



The main network is configured to provide a supplementary output which is only used during training. A sub-task may be used to define the target for this supplementary output. By forcing the network to perform the sub-task in addition to the main task, the learning of the main task may be improved. It is hoped that the additional error terms introduced by the second output will influence the learned function of hidden units in such a way as to assist in the performance of the main task. This method of training using dual outputs is shown in the diagram above.

7.4.1.6 M5 Embedded Sub-Network



This implementation method requires provision of a sub-network which is first trained and then incorporated into the main network. This is shown diagrammatically above for a NNWF. The sub-network must be allocated its own targets during the pre-training. This method can be considered as a generalisation of method M3 which uses a sub-network but requires that it be at the front of the main network.

A sub-network may either process inputs directly or may process the outputs from a number of other sub-nets or sub-tasks. When the sub-network is incorporated into the main network its weights can either be frozen, allowed to adapt slowly or treated in the same way as the other network weights. One advantage of using a sub-network for a sub-task is that errors can be back-

propagated through a sub-network in the same way as they are back-propagated through other sections of the main network. Use of the BP algorithm does not require that all weights be modifiable, only that the delta term of equation 2.7 can be calculated. Each neuron has a delta term which represents the portion of error which has been back-propagated to the neuron and back through its squashing function. The particular package used for the experimentation described in this thesis was Aspirin/Migraines [Leighton 1991]. This did not allow for implementation of the second option of a sub-network with slower weight changes than the main network, therefor this aspect of embedded sub-networks has not been investigated. A sub-network with either fixed weights or weights which change at the same rate as the others in the main network was implemented using Aspirin/Migraines. An example is given in the next chapter [§8.2.1].

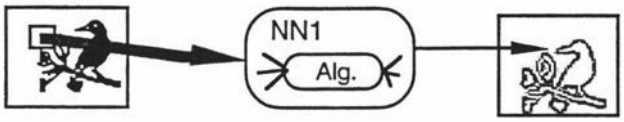
The inputs for the sub-network may either be taken from a scanned window or if non-image operations are being learnt from a file. In a similar way the targets for the sub-network may also be taken from a target image or from a file. For example, if the sub-task being learnt is the logical AND operation the inputs and target would have to initially be provided from a file. Later when the sub-network is incorporated into the main network the inputs would come from the scanned window, or possibly from other sub-networks which had already processed the window inputs.

When this method is used to implement the pre-learning of a useful sub-task a sub-network is trained to perform the sub-task. The advantage of using a sub-network is that it can continue to adapt after incorporation into the final network. If the sub-task can only be defined in an approximate form this can be used for the pre-training. The exact function of the sub-network can be refined by further training once it is incorporated into the main network. This is particularly useful when the sub-task is defined by a rule which only provides an approximate definition of the required sub-task.

In human training, a cue is a stimulus which already elicits the required response. In using this concept in training ANNs, a possible cue is first identified. It is then necessary to incorporate a sub-task into the network which can elicit the required response when presented with the cue. One method of doing this is to train a sub-network for the sub-task(i.e. cue -> required output). Once this has been achieved the cue can form a new or additional target for training the main network. For example, improved performance of the NNWF as a Marr Hildreth

edge operator was obtained by the training of sub-networks. This is described in chapter five [§5.4.1].

7.4.1.7 M6 Embedded Sub Algorithm



This method is very similar to method M5 in which a sub-network is used within the main network. However, in this case, it is an algorithm rather than a sub-network, which is embedded in the network. The constraints on a unit which is to be embedded in a neural network require careful consideration as they will limit the types of functions or algorithms which can be used.

Consider placing a functional unit with multiple inputs and a single output in various positions within a back propagation network. If the unit is placed between the inputs and the first hidden layer or in the first hidden layer, there are no special constraints as it does not take part in the weight modification calculations. All other positions can be considered as equivalent to each other in terms of constraints. For the forward pass, the function can be evaluated at the same stage at which the output of a neuron in the same position would have been evaluated. The output value from the function can then be used in exactly the same way as the output from a neuron would have been used. In the reverse pass, for back propagation of error, replacement of a neuron by a function block can be considered in the following manner; In order to back-propagate the error to the previous layer to calculate weight changes the partial derivatives of the function output with respect to each input is needed. These partial derivatives will need to be bounded and must not be continuously zero over any range of the input values.

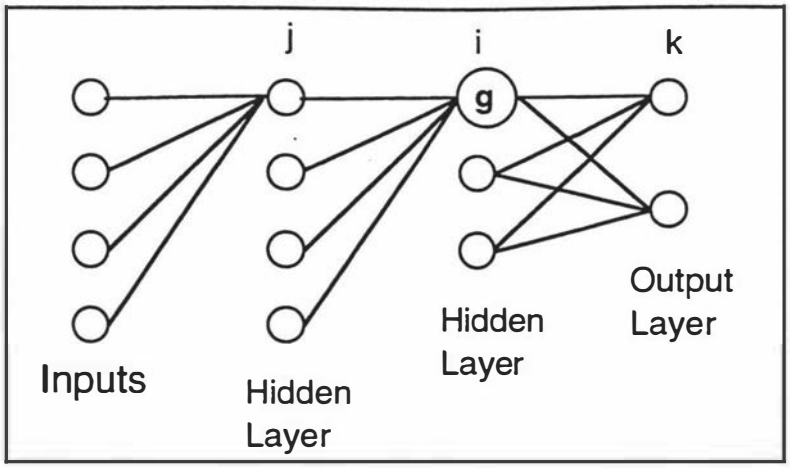


Figure 7.3: Use of an embedded function

The replacement of a single neuron by a general function g which receives inputs from several of the units in the previous layer and propagates an output to the following layer in the same manner as other neurons in the layer will only affect calculation of delta terms. Referring to Figure 7.3 above in which the top neuron in a hidden layer i has been replaced by a general function g. The delta term for the function will have the same form as in equation 2.7.

For a function g in layer i $$\delta_g = \left(\sum_k w_{kg}\delta_k\right) g'(\mathbf{x}) \quad \text{.....7.1}$$

Where the input to the function g is represented as vector **x** this will consist of outputs from various neurons in the previous layer j. Which in turn will be calculated from the neurons in the previous layer. The second term g' in 7.1 is required so that the functions delta term can be evaluated and used to calculate delta terms and hence weight changes in the previous layer. The function g plays no direct part in calculating weight changes in the layer in which it is contained or subsequent layers.

Delta terms in the previous layer j have the form

For hidden unit j $$\delta_j = \left(\sum_i w_{ij}\delta_i\right) f'(a_j) \quad \text{.....7.2}$$

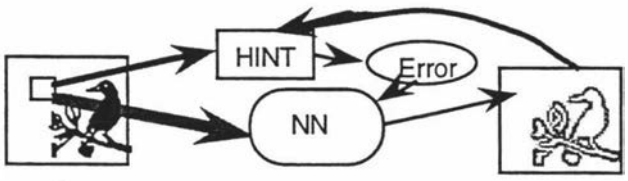
The summation is over all the units in layer i which contains the special unit g. The term inside the summation when connection is to the function g in layer i will have the form

$$w_{gj}\delta_g = w_{gj}\left(\sum_k w_{kg}\delta_k\right)\frac{\partial g}{\partial o_j} \quad \text{.....7.3}$$

as the derivative must be a partial derivative with respect to the input under consideration with the rest of **x** constant. The weight term outside the summation can be taken as one but the delta term for g will be different for each connection to it as a different partial derivative is involved. This means a set of delta terms will have to be calculated for unit g, one for each neuron connecting to it from the previous layer.

Thus a function to be used in place of a neuron in a back propagation network must be an analytic function for which the partial derivative with respect to each particular input, given a set of fixed inputs for the other inputs, is known or can be calculated. No experimental work has been performed on the use of embedded algorithms of this form. The description here is included for completeness, this is an area which warrants further work.

7.4.1.8 M7 Direct Use of Error or Hints



The learning algorithm used in the NNWF requires an error signal which is normally generated from a comparison between the actual output and a target value. There is no requirement that the error to be back propagated be generated in this way. This realization leads to consideration of other ways in which the error signal may be generated. Once generated it can either be used instead of, or in conjunction with the target-output error. The process is shown diagrammatically above for the NNWF. An error signal is derived from the scanning input window and/or the network output by means of a hint or general rule. No target image is used in this process. A rule may also make use of the network inputs and output at two successive positions. For example, a rule or hint may be that for a particular task the operation should be unaffected by reflection of the input image in the horizontal direction. In this case the input window could be processed twice, the second time with a reflection of input window values in the horizontal direction. The difference between the two outputs produced could then be used as an additional error measure. A large

difference would indicate that the final network would not perform in a way which was unaffected by reflection in the horizontal direction.

As an example that makes use of successive positions consider the following situation. A NNWF may be being used to indicate optical flow in all positions in pairs or sequences of images. For particular situations it may be known that no two adjacent pixel positions can be moving in opposite directions. In this case an additional error term could be computed for each pair of adjacent window positions. This term could penalize pairs which had opposite directions for calculated flow.

The direct use of errors or hints is most useful for implementing the fourth heuristic which uses rules to improve training. It can also be used for learning sub-tasks or simpler, similar tasks by the provision of error feedback which is not derived from a difference between actual output and target values.

7.4.1.9 M8 Structural Changes to the Network

An analysis of some tasks may indicate constraints which could be imposed on the structure of the network. For example, if the window filter is being used to detect edges we would expect its response to be independent of object orientation. This implies that the network weights should be symmetrical with respect to window orientation. This constraint on the weights is best built in to the network as it provides fewer degrees of freedom and hence should speed up learning. Weight constraints of this form can be incorporated in two ways. Either the weights which should match can be averaged after each training pair presentation or the network can be implemented in such a way that the weights are actually shared [Driankov et al. 1993].

Other structural change which may be indicated by aspects of a particular task are the number of hidden units which are likely to be required, the connectivity required between parts of the network or the number of hidden layers.

7.4.2 Methods for implementing sub-tasks

The three main methods of implementing sub-tasks are;

- M3 Dual Inputs
- M5 Embedded sub-network
- M6 Embedded sub algorithm

These methods have already been described with reference to sub-tasks, however in certain situations three of the other methods can also be used for learning sub-tasks. These are;

- M1 Sequence. This can only be used when the sub-task forms part of a simple sequence in which the output from one task provides all of the inputs for the next task.
- M2 Pre-training. Surprisingly this method is also sometimes successful. This may be due to the sub-task really being a simpler similar task. Alternatively it may be due to the training for the sub task forming useful hidden units which can later be used in performance of the main task.
- M4 Dual Outputs. This depends on effects produced by the additional error terms provided by the second output. This method of implementing a sub-task will only help if the additional error terms cause the formation of more "useful" hidden units.

7.4.3 Methods for Simpler Similar Tasks

Consider the methods introduced in the previous section as they should be applied to the learning of simpler similar tasks. The main method used is that of pre-training. The pre-training on a simpler similar task provides a means of getting the network into approximately the right area in weight space. Once this has been achieved then the network may converge on the final solution with less danger of being caught in local minima. For difficult tasks it may be necessary to form a sequence of simpler similar tasks, each incorporating more of the complexities of the final task. It would seem likely that over-training on each sub-task should particularly be avoided in such sequences, although no experimental work has been completed to confirm this.

In addition to pre-training, the other methods M3, M4, M5 and M6 can also be used for simpler similar tasks by considering the simpler, similar task as a sub-task. However, this approach will generally sacrifice the advantage of a shaping

progression which is the key reason for training on a simpler task. However methods M3, M4 and M5 could enable a part of the network to perform the simpler, similar task while the extensions or refinements are handled by other parts of the network. The use of an embedded algorithm for a simpler similar task (M6) while feasible seems unlikely to provide any additional benefit.

7.4.4 Methods for using Rules

This section discusses the application of the methods introduced above for improving training by the use of rules.

The most appropriate method of implementing the *use of rules* heuristic depends on the way in which the rule is being used:

- When the rule is being used to produce additional examples method M2 for pre-training should be used.
- When using a rule to define a sub-task, all the methods applicable to the use of sub-tasks can be used. M3 Dual Inputs, M4 Dual Outputs and M5 Embedded sub-networks being the additions here. In addition, as the rule defines a sub-task an embedded sub-algorithm may be appropriate.
- When the rule is used to describe a simpler, similar task for pre-training the main method will be M2. However, M3 Dual Inputs, M4 Dual Outputs, M5 Embedded Sub-Network and M6 Embedded Sub-Algorithm may also be used.
- When the rule is used to directly constrain network structure, then method M8 is naturally appropriate. This method involves the incorporation of the rule or hint into the actual network structure. The network is then custom designed rather than general purpose. The constraint is not learned, it is a fixed part of the system.
- When a rule is being used to provide a direct input of error method M7 is appropriate.

7.5 Structured Training

Experienced teachers are aware of the importance of the selection of the initial examples used to introduce a new concept or method. This heuristic is based on this observation and incorporates related techniques suggested by the instructional design theories discussed in the previous chapter.

Structured training involves the selection and sequencing of examples or the selection and sequencing of the application of the other four heuristics. Incorporated are the ideas of elaboration, shaping, incrementing and the use of matching.

- Shaping consists of learning a series of approximations to the final task. Incrementing involves the learning of a series of sub-tasks which together will perform the desired task.

- Elaboration involves the selection of a single concrete example which epitomises the task at the start of training.

- The use of matching involves the presentation of pairs of examples one of which is part of the concept being taught and one which is not. The two examples should be as similar as possible, that is they should match in all ways which do not affect the classification.

In applying these techniques to neural networks either selections and sequences from a normal set of examples can be used or the other heuristics can be used to create additional examples or modifications to the network or its environment to form a structured sequence. The question of when such structured training is likely to be needed with ANNs will require further research.

7.6 Summary & Discussion

A number of heuristics and methods of implementation have been discussed in this chapter. The general aim in using these heuristics is to improve the training process and hence the capability of artificial neural networks to learn to perform particular tasks. The first heuristic considers methods of assessing and possibly changing the difficulty of the required task in terms of stimulus response characteristics. The next three heuristics involve specific ways in which training can be assisted. Surprisingly, perhaps, it was found that a number of general ways of modifying the network or the environment could be used to implement several of the heuristics. For this reason the heuristics and methods are related through a matrix. The last heuristic is independent of the actual network type and considers the more general question of sequencing and selection of the other strategy components.

A few specific points not covered in the previous sections of this chapter can also be made here;

Where digitised images of the physical world are involved, artificial precision can be implied by the process of digitisation. This is particularly relevant in the generation of target images for training. The use of fuzzy targets may merely compensate for this artifact and may therefore be more applicable to real world images than to computer generated images.

Where particular sub-tasks imply a sequence of operations, methods which use a sequence of nets will be more appropriate. In some cases the use of sub-tasks may be the result of the observed human tendency to describe a task as a sequence of operations. In such cases it may be better to implement and learn the task directly as one operation using a single network.

7.7 REFERENCES

page first cited

Abu-Mostafa 1993
Y.S. Abu-Mostafa, *Learning from Hints*, in Proceedings of Third Australian Conference on Neural Networks ACNN'93, Melbourne, Feb 1993, pp37-40 ..... 7.14

Catania 1992
A.C. Catania, Learning, Prentice Hall, London, 1992 ..... 7.11

Cornsweet 1970
T.N. Cornsweet, Visual Perception, Academic Press, New York, 1970 ..... 7.7

Dorner 1983
D. Dorner, *Heuristics and cognition in complex systems*, in Methods of heuristics, Lawrence Erlbaum Associates, New Jersey, 1983 ..... 7.10

Driankov et al. 1993
D. Driankov, H. Hellendoorn, M. Reinfrank, An Introduction to Fuzzy Control, Springer-Verlag, New York, 1993 ..... 7.24

Grandvalet et al. 1991
Y. Grandvalet, S. Canu, D. Meizel, T. Lanlois, *Preconditioning for improving convergence of back propagation algorithm in control applications*, in Intelligent engineering systems through artificial neural networks: proceedings of the Artificial Neural Networks in Engineering Conference, ASME Press, New York, 1991, pp89-94 ..... 7.6

Hrycej 1992
T. Hrycej, *Supporting supervised learning by self-organisation*, Neurocomputing, Vol. 4, No.1-2, 1992, pp17-30 ..... 7.6

Kohonen 1990
Teuvo Kohonen, *The Self-Organizing Map*, Proceedings of the IEEE, Vol. 78 No. 9, Sept 1990, pp1464-1480 ..... 7.6

Le Cun et al. 1989
Y. Le Cun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard , L.D. Jackel, *Back-propagation applied to handwritten zipcode recognition*, Neural Computation, Vol. 14, 1989, pp541-551 ..... 7.13

Leighton 1991
R.R. Leighton, The Aspirin/MIGRAINES Software Tools: User's Manual The MITRE Corporation, 1991 ..... 7.20

Manning 1979
Aubrey Manning, An Introduction to Animal Behaviour, Edward Arnold Publ., London, 1979 .. 7.11

Ngan 1993
P.M. Ngan, The Development of a Visual Language for Image Processing Applications, Thesis, PhD, Massey University, 1993 ..... 7.10

Rumelhart & McCleland 1986
D.E. Rumelhart, J.T. McCleland, Parallel Distributed Processing, Explorations in the Microstructure of Cognition. Three Volume Set, MIT Press, Cambridge MA, 1986, p329 ..... 7.7

Tuck 1993

D. Tuck, Practical Polynomial Expansion of Input Data Can improve Neurocomputing Results, in Artificial Neural Networks and Expert Systems ANNES'93, Dunedin, Nov. 1993, pp42-45..... 7.8

Chapter 8

8. Experiments with an Educational Environment for the NNWF

In this chapter experiments are reported which start to fill in the heuristics-methods matrix proposed for the neural network window filter educational environment. There are two aims here; firstly to test the validity of the training environment concept and secondly to determine if particular methods are more suitable for applying particular heuristics. A complete instructional design theory should not only suggest methods but also indicate when to apply each method or combination of methods. The experiments reported form the start of a development cycle which may eventually produce a complete theory of instructional design for artificial neural networks.

8.1 INTRODUCTION

The work described in this thesis is intended to demonstrate the validity of an approach to improving the performance of the NNWF by using an educational environment. It was not intended to produce or validate a complete theory of instructional design for neural networks. The broad aim of the experimental work reported in this chapter was to determine the validity of the educational training strategies developed in chapter seven. It is beyond the scope of this work to fully test all the combinations of heuristics and methods, however, a representative sample was selected to verify the usefulness of the general approach. In the first series of experiments all the methods of applying the sub-tasks heuristic are tested. In the second series an attempt is made to apply each of the heuristics in turn to a single image processing task.

A suitable set of image processing tasks and sub-tasks had to be identified for these experiments. If the NNWF could learn to perform a task perfectly when presented with a simple set of training examples there would be no opportunity for the use of special methods. On the other hand if the task was insoluble, within the constraints of a window filter, again there would be no opportunity to test the usefulness of special methods. This identification of difficult but possible tasks has proved surprisingly elusive.

To evaluate the usefulness of parts of an educational environment it is necessary to compare the training of NNWF in different situations. This comparison was made on the basis of network performance after a set number of training cycles. Ideally either learning effectiveness or learning efficiency should be measured. That is, how much learning was produced regardless of effort or how much per unit of effort. Both of these are difficult to define or measure. Using training cycles as a measure may give an undue advantage to complex training methods or networks which actually perform more computations in a single training cycle, however the alternative of comparing CPU time for simulations also has a number of dangers; the efficiency with which various different approaches have been programmed will become significant and algorithms which are inherently more parallel may appear at a disadvantage when simulated on a serial computer. For these reasons in the experiments that follow the performance, in terms of RMS error, after a set (adequate) number of cycles is used as a measurable and approximate alternative. The equivalent situation, using the analogy to human

instruction, would be applying a standard test after a set number of tutorial lessons.

The first series of experiments described here (§8.2.1) were used to investigate the use of sub-tasks in the educational environment, first in isolation and then in combination with fuzzy targets. Each of the methods of implementing the sub-task heuristic for the NNWF was considered in turn. Two different sub-tasks for the same main task were investigated. This provided some scope for distinguishing between general effects due to the sub-task implementation method and effects which are specific to a particular sub-task.

In the second series of experiments an attempt was made to bring each of the heuristics to bear on a single real image processing task.

8.2 EXPERIMENTS.

8.2.1 Sub-task Experiments

In these experiments the filter was required to mark the edges of shapes containing a particular texture. The input image used is shown in figure 8.1a below. The task was to mark the edges of objects containing the  texture. This is a contrived task, designed to provide a clear sub-task which could be easily identified and would be likely to assist in the performance of the whole task. During the experiments the top half of the image was used for training and the bottom half for verification and testing. The target for the complete task was generated by using NIH Image as follows: The original image was constructed using simple drawing and shading tools. Then a human operator was asked to rub out all the objects of incorrect texture. The edges of the remaining objects were then found by performing a series of steps; smoothing, thresholding, edge detection and skeletonization. The resulting target is shown in figure 8.1b.

Two sets of experiments were to be carried out on these images; one in which "selection of areas with the correct texture" (upward sloping bars) was the sub-task and one in which "finding edges of all objects" was the sub-task.

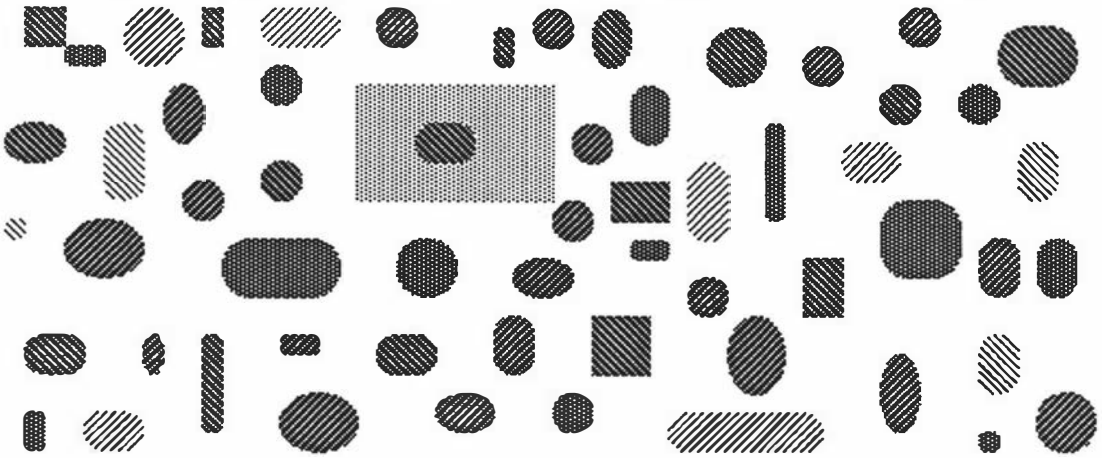
It is important that the performance of the naive NNWF on the whole task is not too good if the task is to be used to investigate special training methods. The first step was to test the NNWF on the complete task. The results of training for 200 epochs with the following network parameters is shown in figure 8.1c and when compared with the target image provides RMS errors of 10 and 22 for the training and verify sets. The RMS errors at 100 epochs was 13 and 23. Further training reduced the error on the training set but not the verification set.

Network Parameters:

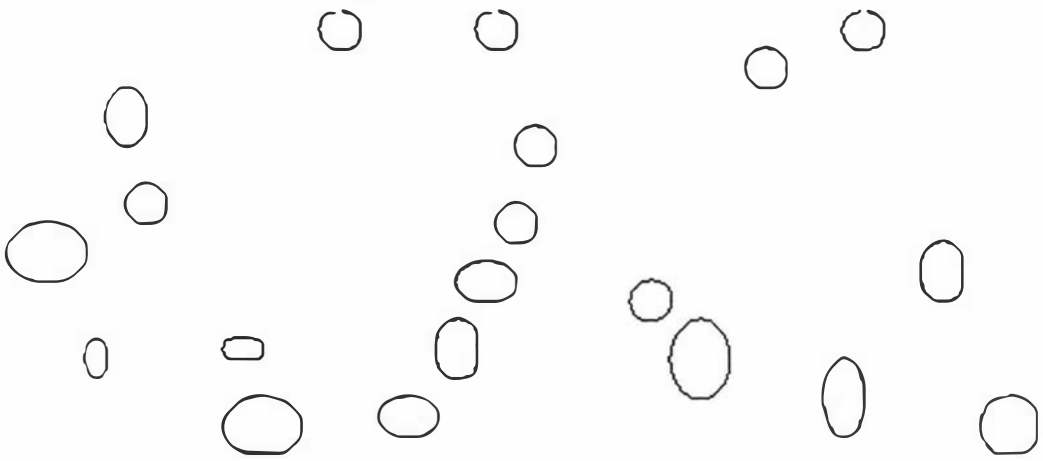
Heuristic and Method	H0M0
Input window size	9 x 9
Neurons in first hidden layer	5
Neurons in second hidden layer	3
Initial Gain	0.01
Momentum	0.9
Minimum for dynamic gain	0.005
Target	tshapes_nsl

Table 8-1 Network Parameters

a) Input image simple shapes and textures generated on a computer.



b) Target image showing just edges of those objects with /// texture



c) NNWF Output when trained using target above

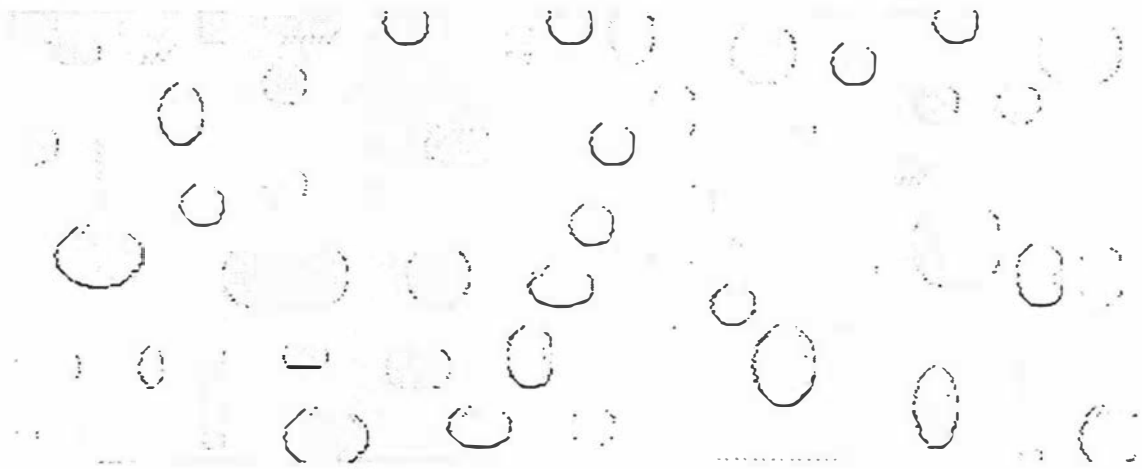
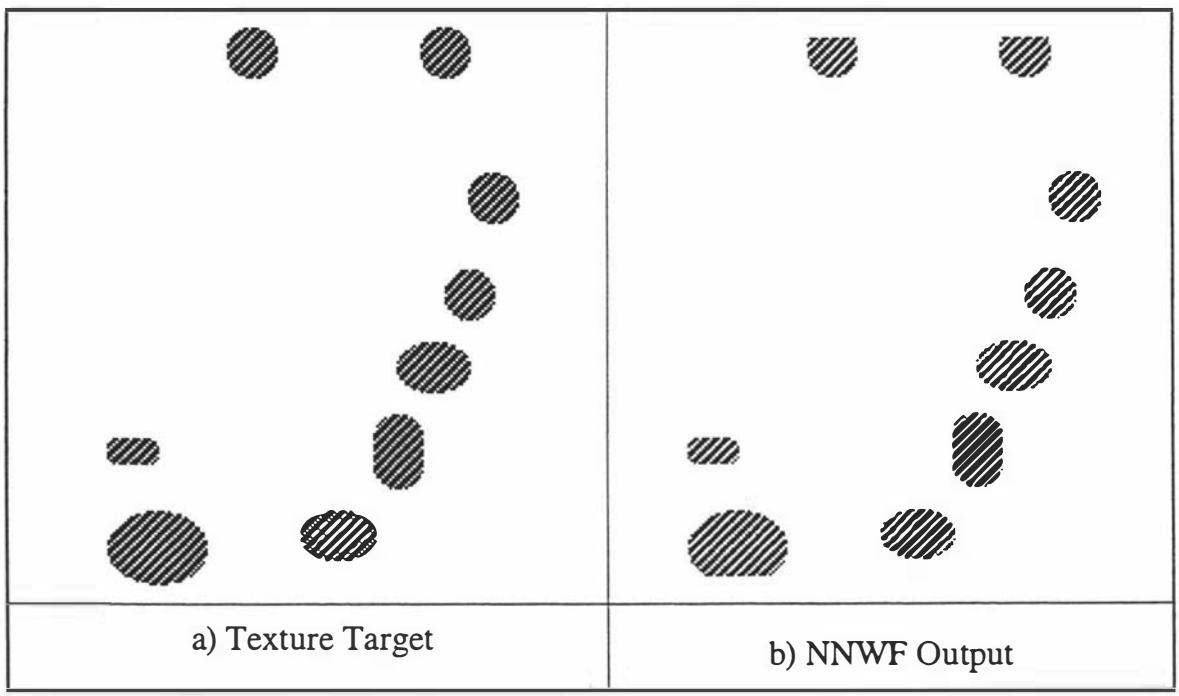


Figure 8.1: Images from textured shapes experiments

8.2.1.1 Experiment 1.1 Texture as a sub-task

To enable the identification of texture to be used as the sub-task a method of indicating areas containing the correct texture is required. The method used was to train the NNWF to "pass through" image areas containing the correct texture. Other areas of the image would be left blank so that the NNWF was acting as a gating filter. A variety of methods of incorporating this sub-task into the main task could then be investigated. The input image used for this sub-task training was part of the image shown in figure 8.1a. A section of the targets for this first sub-task is shown below in figure 8.2a and a section of the output image produced by the NNWF after training the network to perform this function is shown in figure 8.2b. The target image, required during training, was again produced by a human operator. The input image for this training is the left hand side of figure 8.1a. The RMS errors after training for 100 epochs was 0 on the training set and 1 on the verify set. Note the almost perfect performance. The network used had the same form as the main network with 9x9 input window, and network layers containing five, three and one neurons respectively.



a) Texture Target b) NNWF Output

Figure 8.2: Texture as a sub-task

Each of the methods of combining the sub-task were then tried. The results are shown in the figures below. Each figure shows a diagram of the implementation method, performance error figures, a graph of errors during training and a portion of the image produced by the trained network. The RMS error figures for both training and verify sets after 100 and 200 epochs are given. The top half of the image was used as the training set and the bottom half of the image

was used as the verify set. The error figures are derived from the 0-255 grayscale values used in the actual images. The section of output image shown was produced by the NNWF after 200 epochs of training in each case. The graph shows the training and verify errors at the end of each epoch during training, the higher darker line is the verify error. The graphs are all plotted on the same scale to allow comparison. Detailed notes on each of the implementation methods are given below:

8.2.1.1.1 **H0M0 Standard Training Using Examples**

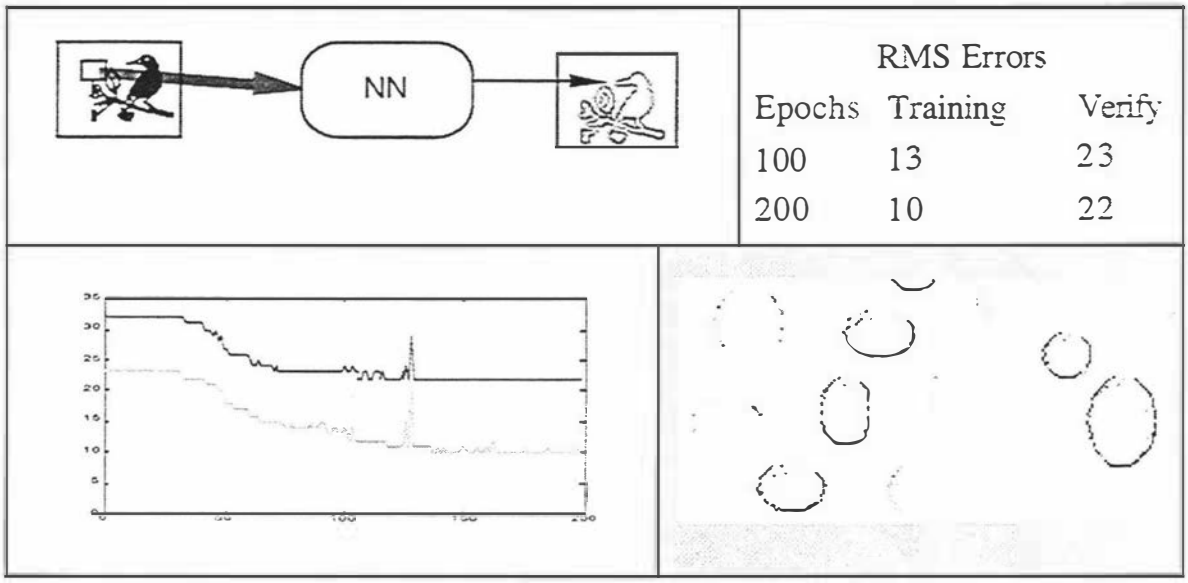


Figure 8.3: H0M0 Standard Training Using Examples

This is just the standard training by examples method as described in the previous chapter and is included to enable comparison with the educational environment methods. In the output image shown above several deficiencies can be seen; firstly the required edges are not complete, secondly spurious edges from other objects with the incorrect texture are also being picked up and there are also a number of incorrect edge points being shown in the interior of the textured objects.

8.2.1.1.2 H2M1 Sub-Task Implemented as a Sequence

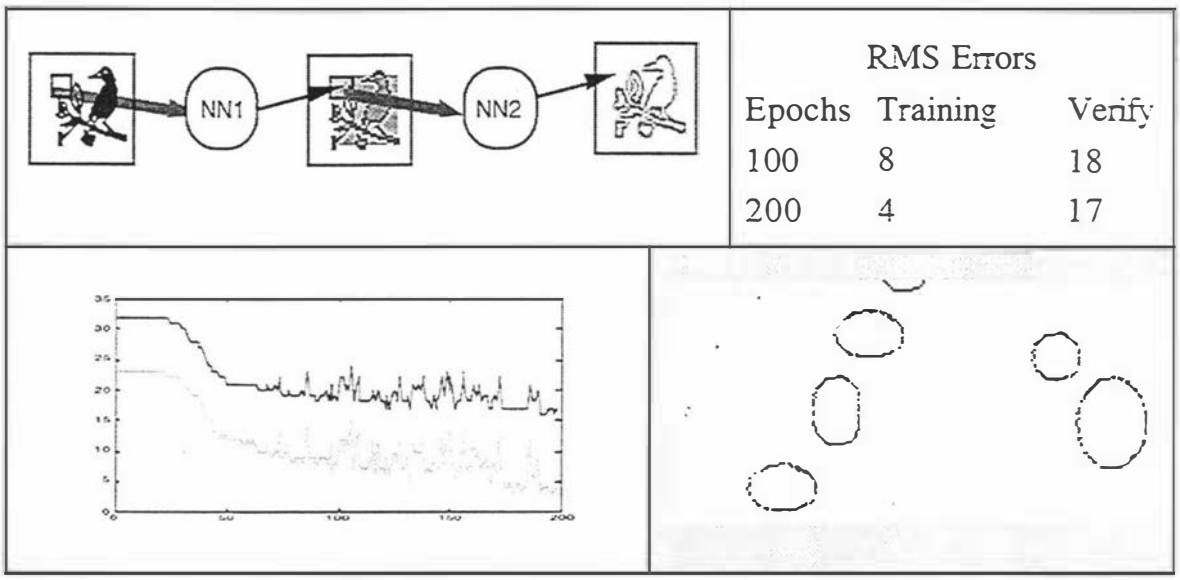


Figure 8.4: H2M1 Sub-Task Implemented as a Sequence

Here two networks are used, the first is trained to perform the chosen sub-task, that is to filter out all regions not containing the correct texture. It produces an image as output. A section of this output is shown in figure 8.2b. This intermediate image is then fed into a second network which is trained to produce the final output. Both networks have the same structure. It should be noted that a sub-task can only be implemented in this way if the final output can be derived from the output of the sub-task alone. It can be seen from the figures given that using this method the performance of the NNWF was much better with almost no error on the training set and a smaller error on the verify set.

8.2.1.1.3 H2M2 Sub-Task Implemented as Pre-training

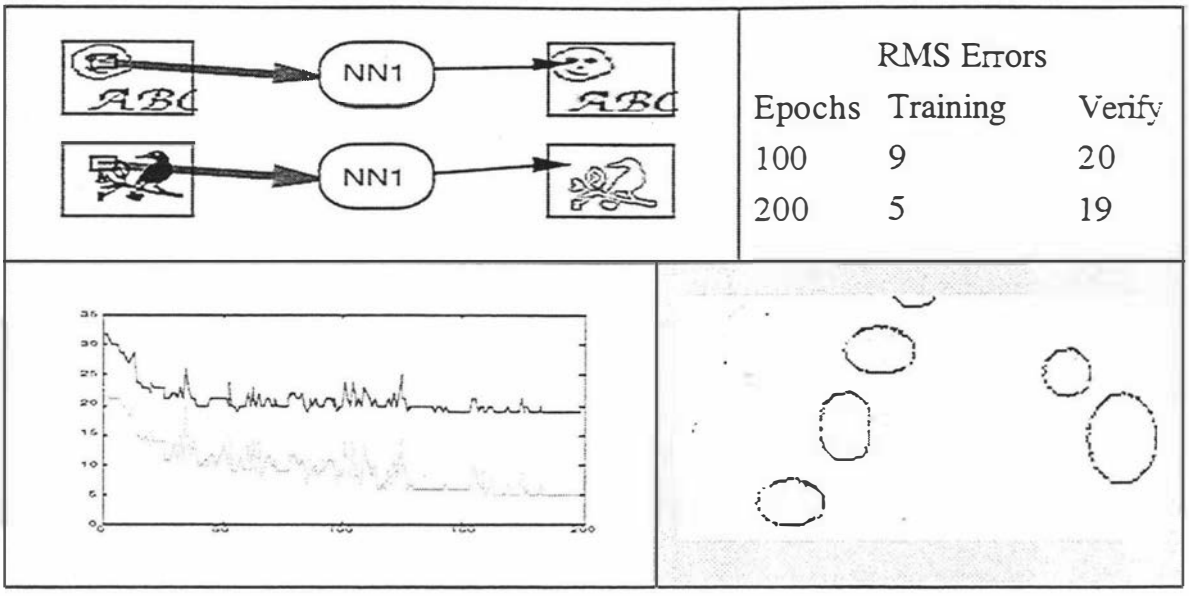


Figure 8.5: H2M2 Sub-Task Implemented as Pre-training

In this method the main network is pre-trained to perform the sub-task and then this network, complete with all its bias and connection weights, is used as the starting point for training on the main task. Thus, pre training is used as an alternative to random selection of initial weights. It was expected that this would only help if the sub-task was also a simpler similar task. If the sub-task has nothing in common with the main task starting training on the main task in this area of weight space is unlikely to have anything other than random advantages. On the other hand if the sub-task can also be considered a simpler similar task then the position in weight space could be expected to be "closer" to the solution point for the main task than a random position.

While the experiment using this implementation of a sub-task did not perform as well as the previous one, it still provided a significant improvement. This could suggest that "passing through the correct texture" can, at least to some extent, be considered a simpler, similar task to "marking the edges of objects containing the correct texture".

8.2.1.1.4 H2M3 Sub-Task Implemented as Dual Inputs

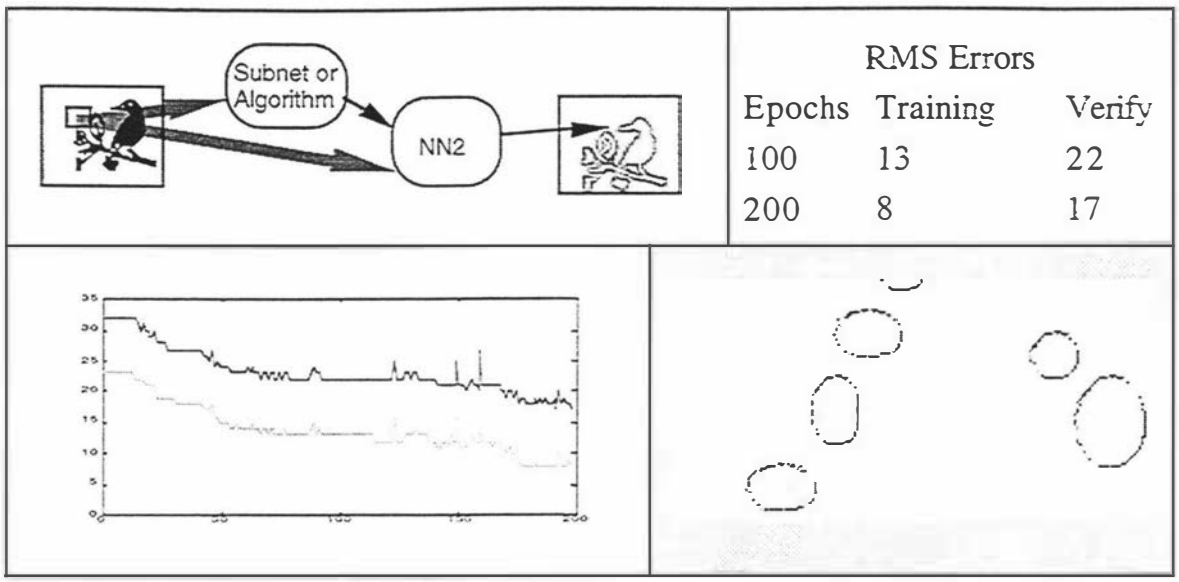


Figure 8.6: H2M3 Sub-Task Implemented as Dual Inputs

This is the first of three main methods of implementing a sub-task. In this example a sub-network rather than a sub-algorithm was used. The sub-network used is described at the start of this section with its output shown in figure 8.2b. This output provides a second input image which is scanned by the NNWF at the same time as the original input image is being scanned. This implementation method has also provided a significant improvement in performance. In comparison with methods M1 and M2 the improved performance appeared later in the sequence of training cycles.

8.2.1.1.5 H2M4 Sub-Task Implemented using Dual Outputs

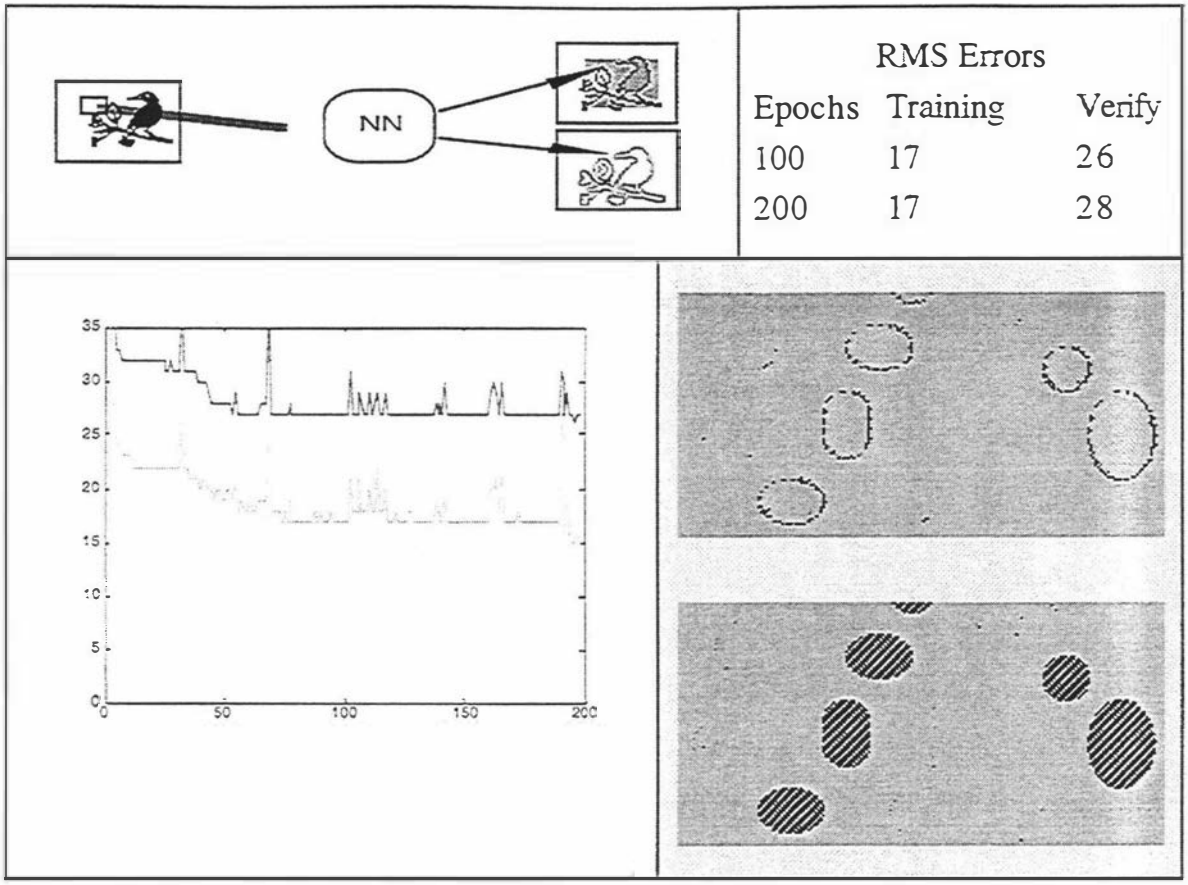


Figure 8.7: H2M4 Sub-Task Implemented using Dual Outputs

This implementation is based on the notion that the outputs required for the sub-task may "pull" some of the hidden units into a position in which they will be performing useful functions for the solution of the main task.

The two outputs produced by the trained NNWF in this experiment are shown above. The RMS errors given above are calculated from final NNWF outputs. These results are worse than those from the standard training without the use of a sub-task. This suggests that in this case rather than helping to pull hidden units into useful regions the second output for the sub-task actually moved hidden units away from the required function.

8.2.1.1.6 H2M5 Sub-Task Implemented as an Embedded Sub-network

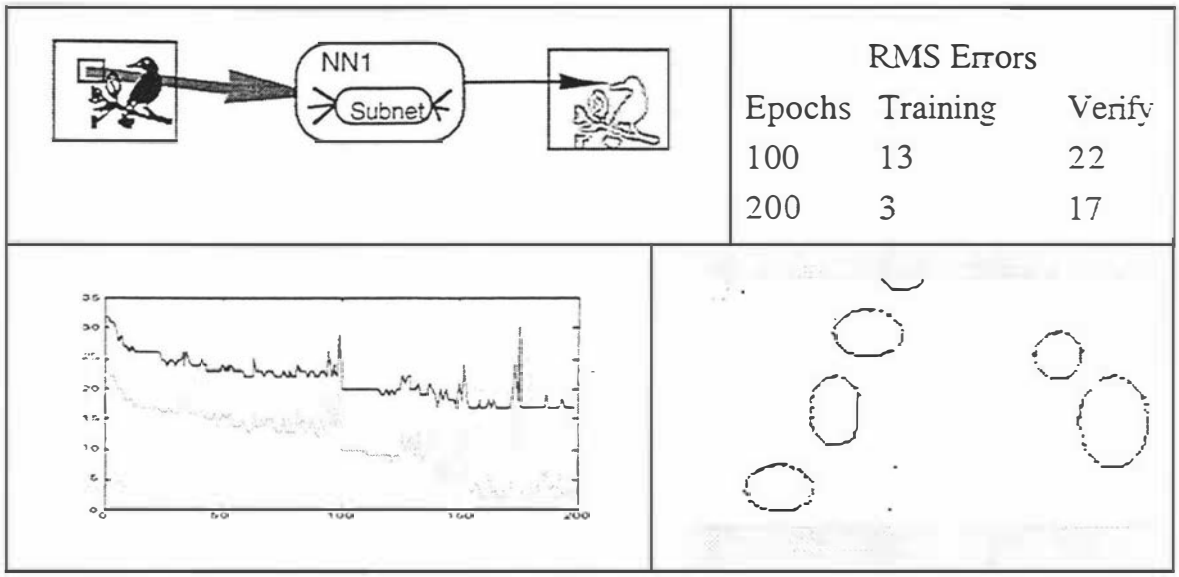


Figure 8.8: H2M5 Sub-Task Implemented as an Embedded Sub-network

This is the second of the three main methods of implementing a sub-task. It involves the training of a sub-network as described above and then insertion of this network into a larger network and continuation of training. The structure used in this experiment is shown below in figure 8.9. The labels S1, S2 and S3 refer to the three layers of the sub-network. These were initially trained as a NNWF to "pass through" areas of the correct texture. These nodes were then incorporated into the network in figure 8.9. The other layers in this network started with random weights and received inputs from the sub-network layers as well as lower layers of the main network. The suffix on each layer indicates how many neurons are contained in that layer. Each arrow indicates a connection from all neurons in one box to all neurons in the other box.

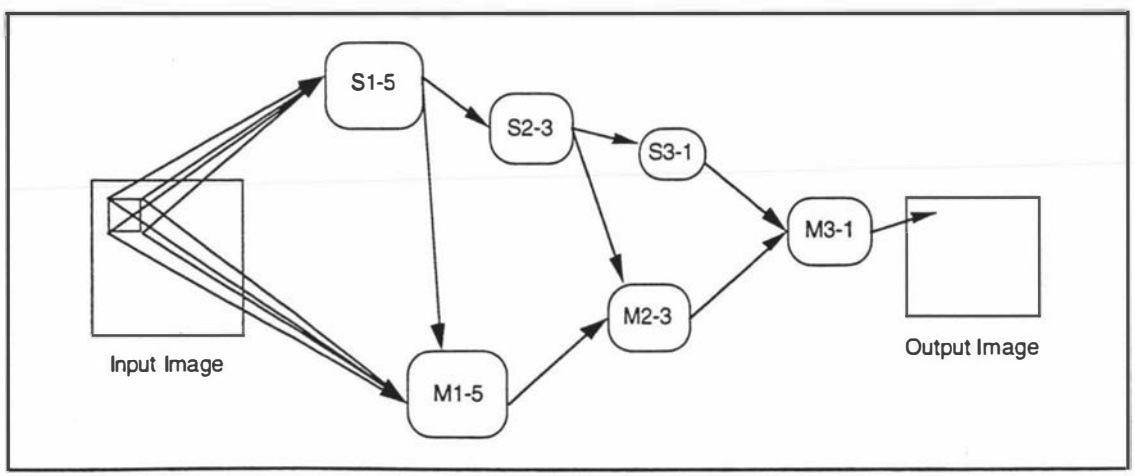


Figure 8.9: Embedded sub-network

The results of the experiment using this implementation of a sub-task are shown above in figure 8.8. Again a significant improvement in performance was produced when compared with M0 training.

8.2.1.1.7 H2M6 Sub-Task Implemented as an Embedded Sub-algorithm

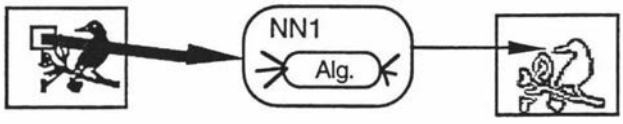


Figure 8.10: H2M6 Sub-Task Implemented as an Embedded Sub-algorithm

An algorithm for performing the sub-task was not developed so this implementation method could not be investigated using this task, sub-task combination. In order to test this implementation method it would be necessary to develop an algorithmic method of implementing the sub-task which met the requirements discussed in section 7.4.1.7 of chapter seven. As this would be a task of considerable difficulty, this was left as an area for possible future work.

8.2.1.1.8 H1 Fuzzy Targets

Experiment 1.1 was then repeated with a smoothed target, a section of which is shown in figure 8.11a. A comparison of the various methods of implementing the texture sub-task, with and without a blurred target are given in table 8.2 below. A section of the output from the NNWF when trained for 200 epochs using method M3 and a smoothed target is shown in figure 8.11b. It is clear from the results shown in table 8.2 that while improvements were made using the sub-task in isolation even greater gains were obtained by combining these two approaches

Method	**Sub-task Heuristic**		**Sub-Task Heuristic with blurred target**	
	RMS error on training set	RMS error on verify set	RMS error on training set	RMS error on verify set
M0 Examples	13 - 10	23 - 22	15 - 14	23 - 22
M1 Sequence	8 - 4	18 - **17**	7 - 7	10 - 10
M2 Pre-training	9 - 5	20 - 19	12 - 10	17 - 16
M3 Dual Inputs	13 - 8	22 - **17**	5 - 4	**9 - 8**
M4 Dual Outputs	17 - 17	26 - 28	6 - 6	12 - 12
M5 Embedded Sub-network	13 - 3	22 - **17**	6 - 5	13 - 12

Table 8-2 Comparative results for texture sub-task

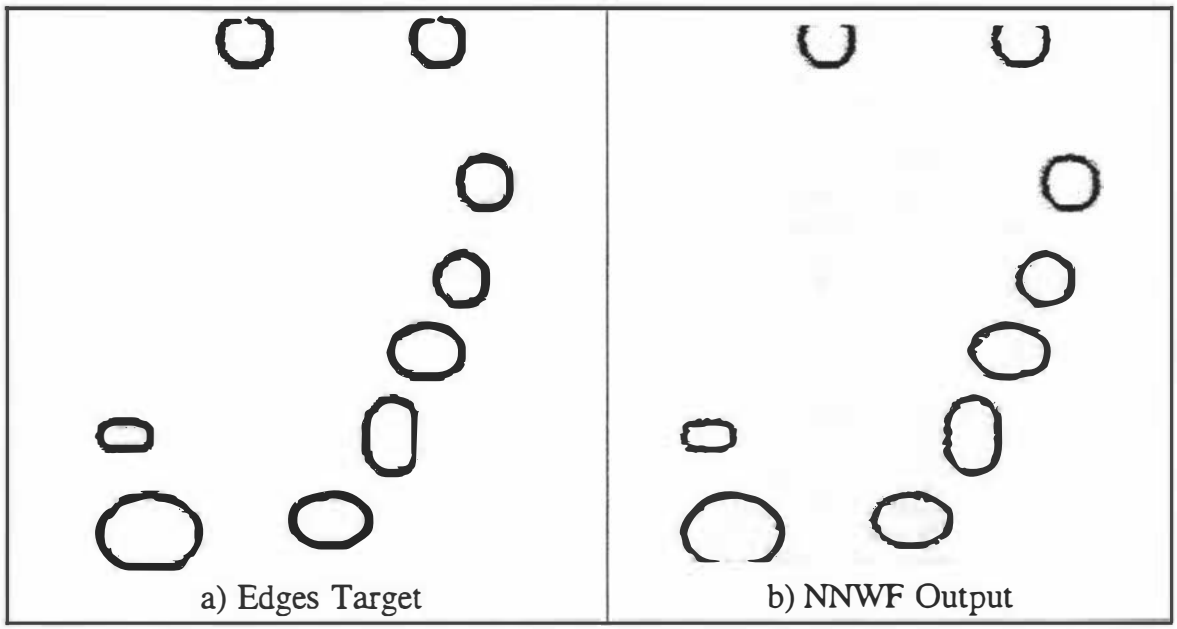


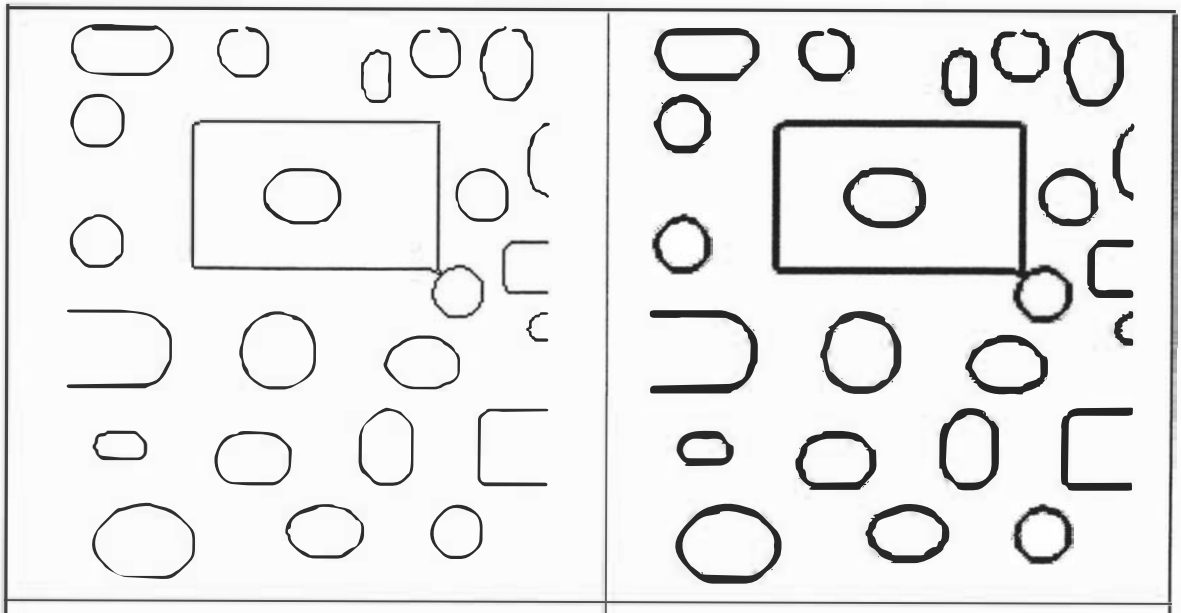
Figure 8.11: Smoothed Target and Best result from NNWF

8.2.1.2 Experiment 1.2 Edges as a sub-task

Similar experiments were then performed using the detection of edges of all objects as a sub-task. A section of the target image for these experiments is shown in figure 8.12a below. Again this was produced by hand using a series of processing steps within NIH Image. A section of the output produced by a trained sub-network is shown in figure 8.13b. A network with the same parameters as for the other sub-task was used. After training for 100 epochs the network produced RMS errors of 32 on the training set and 29 on the verify set. The target for the sub-task both smoothed and un-smoothed is shown in figure 8.12 below. This sub-task does not retain sufficient information in its output image to allow the performance of the main task so cannot be used successfully as a sequence.

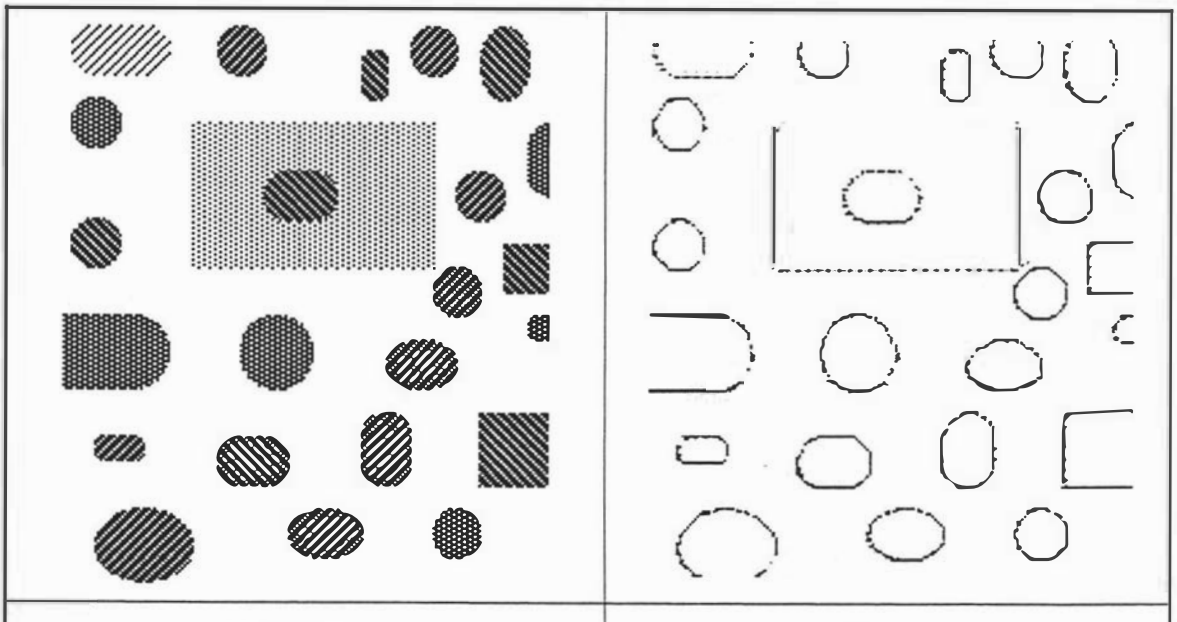
The results of the experiments which made use of this sub-task are shown in table 8.3 below. Note that despite the less than perfect performance on the sub-task its use to augment training still provided significant improvements in performance of the main task. An improved performance was achieved using methods M3 (dual inputs) and M5 (embedded sub-network). As expected its use as a sequence did not provide any improvement. It is interesting to note that it provided no benefit when used in pre-training(M2) or with dual outputs(M4). This is presumably because the hidden nodes which are required for the sub-task are not helpful in performing the main task and the sub-task is not a simpler similar task.

This sub-task was implemented as an algorithm so could in principle allow the use of method M6 (embedded sub-algorithm), however, as the algorithm is not differentiable it cannot be incorporated into a neural network except using method M3 where it is used to provide additional inputs to the main network.



a) Edges Target | b) Smoothed Target

Figure 8.12: Target image for edges sub-task



a) NNWF Input | b) NNWF Output

Figure 8.13: Target image for edges sub-task

Method	Sub-task Heuristic		Sub-Task Heuristic with blurred target	
	RMS error on training set	RMS error on verify set	RMS error on training set	RMS error on verify set
M0 Examples	13 - 10	23 - 22	15 - 14	23 - 22
M1 Sequence	35 - 35	35 - 35	25 - 25	33 - 33
M2 Pre-training	12 - 8	23 - 21	15 - 13	23 - 22
M3 Dual Inputs	5 - 1	17 - 16	8 - 6	16 - 15
M4 Dual Outputs	43 - 42	43 - 43	38 - 36	40 - 40
M5 Embedded Sub-network	5 - 4	18 - 18	5 - 5	**11 - 11**

Table 8-3 Comparative Results for Edges Sub-task

8.2.1.3 Discussion of Sub-task experiments

In the first experiment an error was inadvertently included in the target image for the whole task. During the process of producing the target image by hand two of the correct objects had been eliminated. An interesting observation can be made when the results from this experiment is compared with results of a second set in which this error was corrected. During the initial training the inclusion of some incorrect examples in the training set had little effect on learning rate or performance on the verify set. It was interesting to note that the NNWF still found the edges missed by the human operator. However, once the additional training methods were brought to bear and NNWF performance was nearing perfection, inclusion of the incorrect examples had the effect of limiting the final performance level. This seems quite in keeping with the effect which would be expected from a teacher inadvertently providing some incorrect examples for students; in the early stages of skill acquisition the errors will have little effect, however once performance is near optimal continuing inclusion of incorrect examples will have a more dramatic effect and is quite likely to limit final performance levels.

8.2.2 Finding Tree edges

In this series of experiments the NNWF was used to identify the edges of trees in photographs of pine trees. This was used as an example of a complex task on which a range of educational heuristics could be tested. The aim of the experiments was to determine whether the five heuristics suggested in chapter seven could be applied with advantage to a complex real world task. Just as a teacher would not normally try to apply every tenet of an educational theory on a single pupil and a single lesson we would not normally attempt to apply all of the five heuristics at once. However as the intention here is to explore the use of the set of heuristics experiments relating to each of the five heuristics are discussed in turn in the following sections. The five heuristics are repeated here for reference.

1. The consideration of stimulus response characteristics
2. The use of sub-tasks
3. The use of simpler-similar tasks
4. The use of rules
5. The use of structured training techniques

Before discussing the experiments which make use of the above heuristics it is appropriate to carefully consider the nature of this task and look at both how well it can be performed by other edge filters and how well it can be performed by the NNWF using standard (M0) training. An example image is shown in Figure 8.14. As in many real world situations the task definition was initially a little vague. It is the edges of the trees in the foreground, particularly the central tree which are really of interest. In this sense and ideal output would consist of two continuous lines, one down either side of each such tree. The questions which might be asked to enable filter performance to be evaluated are: What about the edges of other trees? Does it matter if they are also extracted? What exactly is the foreground? How thick should the indicated edges be and when the actual edge position falls between pixel positions how should this be indicated? Can discontinuities be tolerated?

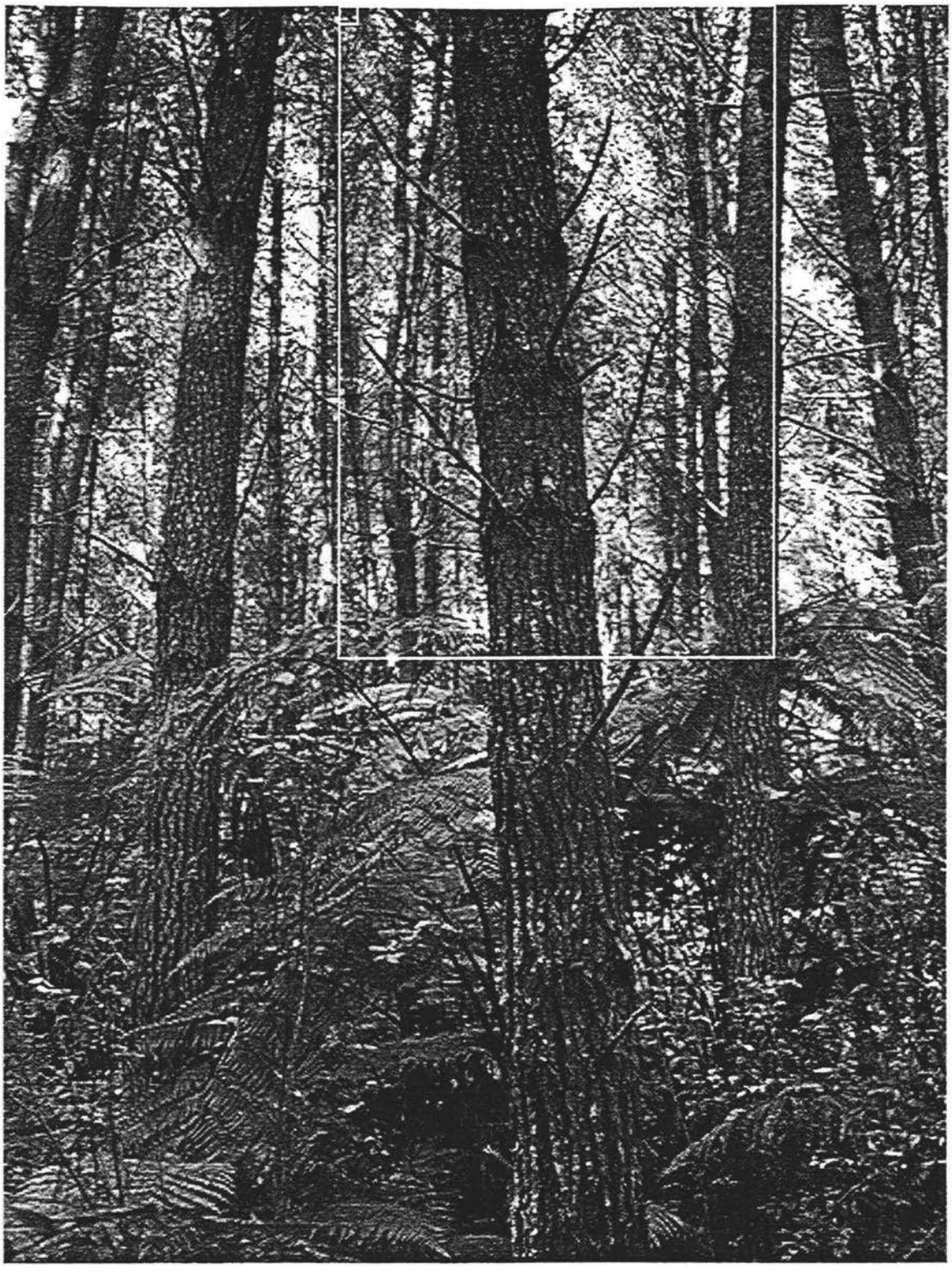


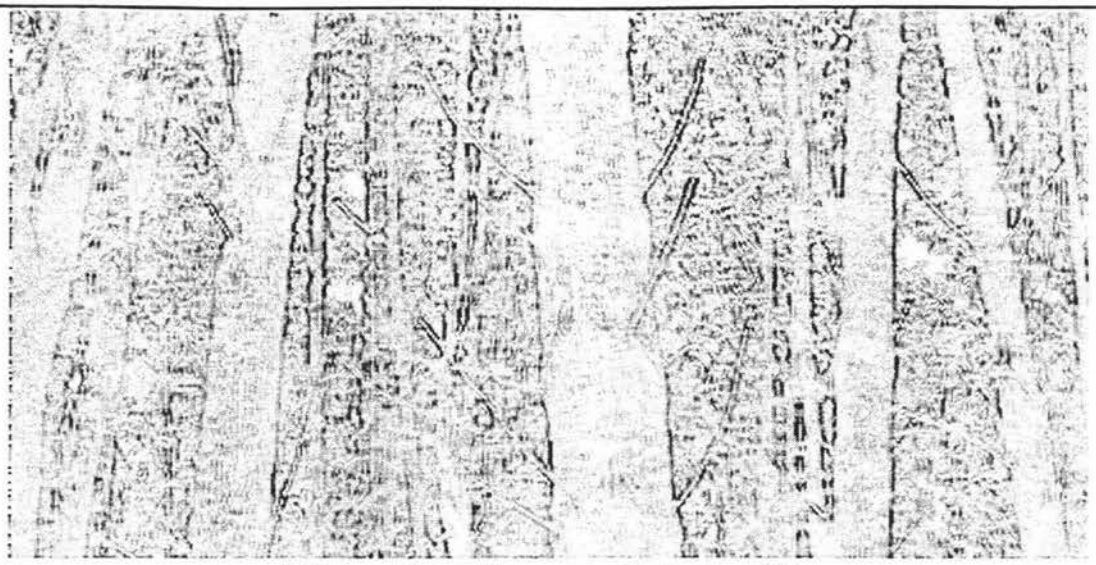
Figure 8.14: Image from Pine Forest

The extent of the training sub-image is shown by the large white square. The size of a 9x9 input window is shown by the small white square.

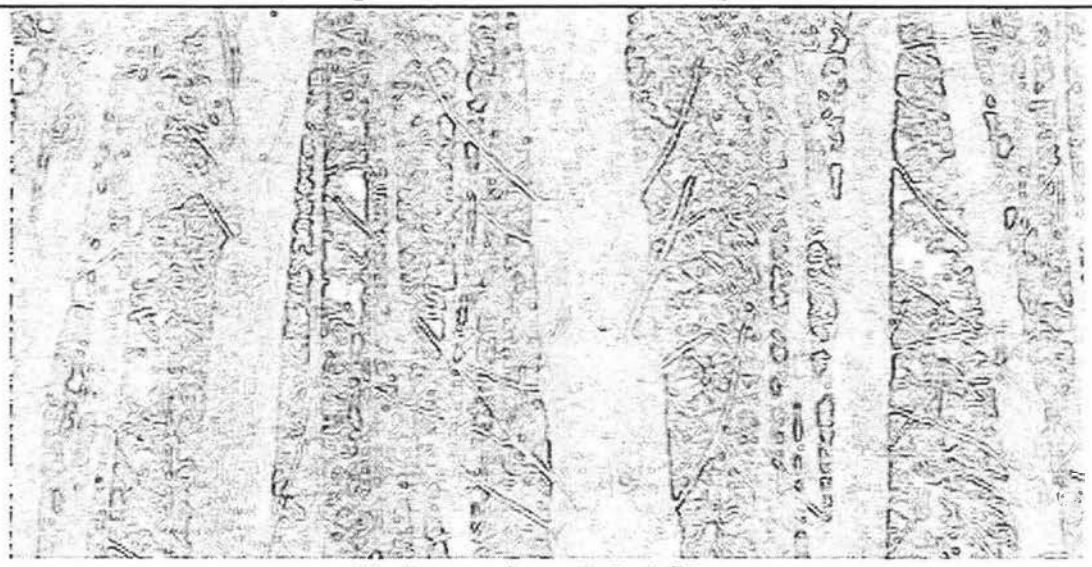
A window filter may either be used to enhance an image for later viewing or as a pre-cursor to some form of image analysis. In this case the analysis which was to be carried out was the measurement of the shape of the stems of the trees in the

foreground. For this reason the required output in this case is one containing a number of edge segments from the trees in the foreground from which it will be relatively easy to determine the stem shapes. As the actual method of doing this will depend on the type of output which can be produced by the edge filter, the target cannot be more precisely described. The exact target definition will depend on what can be achieved and may change as the filter construction proceeds. This is not an unusual situation in image processing.

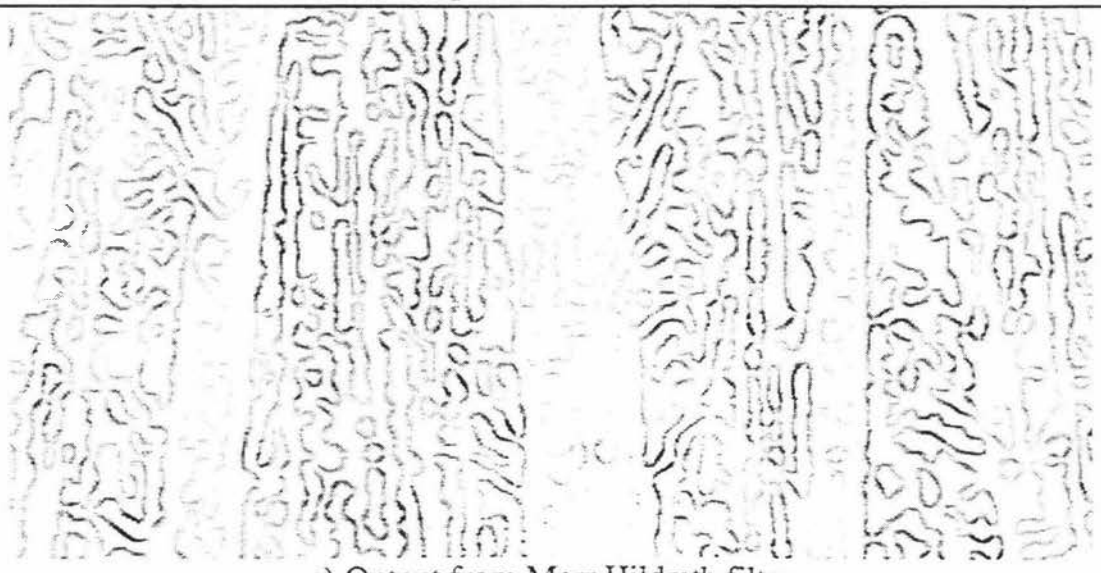
In Figure 8.15 the results of processing a section of the image with the following standard filters are shown: a)A 3x3 linear vertical edge filter. b) A 3x3 Sobel filter. c) A Marr-Hildreth filter with a window size of 23 and a sigma of 3. While these all appear to provide an indication of the positions of the required edges a considerable number of irrelevant edges are also indicated. Even with careful control of filter parameters and thresholding these spurious edges cannot be eliminated. This implies the task is not straightforward and that the NNWF is if anything only likely to produce improved rather than perfect results. For comparison a section of one of the best images produced by the NNWF is shown in Figure 8.16.



a) Output from linear vertical edge filter.



b) Output from Sobel filter.



c) Output from Marr Hildreth filter.

Figure 8.15: Alternative edge filters

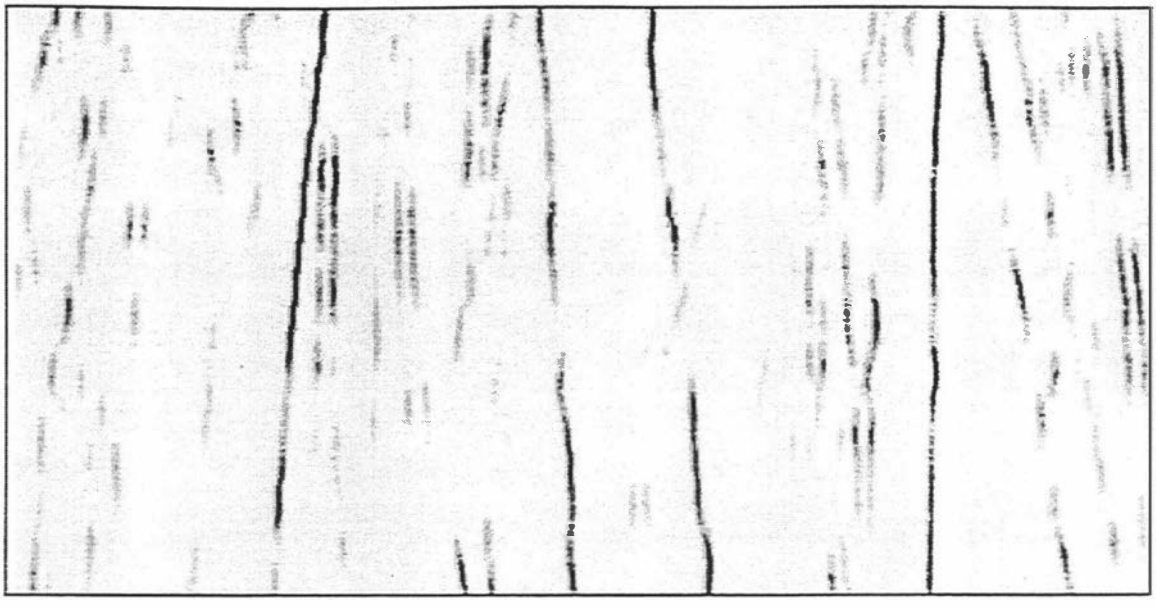


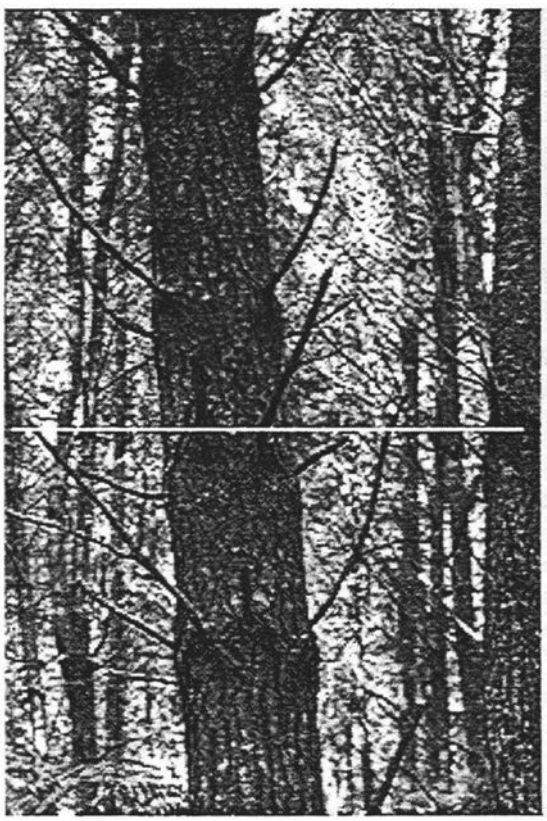
Figure 8.16: Output from trained NNWF

A number of different network sizes and structures were used for these experiments but the base configuration was as shown in Table 8-4 below. This network configuration was chosen after trying smaller and larger networks and smaller and larger window sizes. While an exhaustive search for optimal network size was not undertaken this structure was chosen to ensure network performance, at least for M0 training, was not being limited by the network structure. Variations from this network structure will be indicated in the text. Each trial consisted of one hundred training epochs.

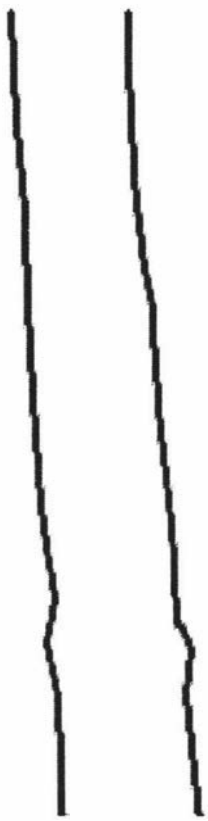
Heuristic and Method	H0M0
Input window size	9 x 9
Neurons in first hidden layer	5
Neurons in second hidden layer	3
Initial Gain	0.001
Momentum	0.9
Minimum for dynamic gain	0.0005
Input Image	cda22

Table 8-4: Network Parameters

As discussed earlier a task must be possible but not too easy if it is to be used to test enhancements to the educational environment of the network. In tasks such as this it is generally impossible to determine what is the best possible result. So the answer to the question, Is the task possible for a window filter? is difficult to answer. An initial test with standard training showed partial success on the task. The target used for training was created by hand using simple line drawing tools and is shown in Figure 8.17b. The output from the trained filter, using standard (M0) training, is shown in Figure 8.19. By adding the filter output to the input image the position of the found edges can be seen more clearly. The partial success of the NNWF provided some justification for trying the various educational heuristics to see if improvements in performance could be achieved.



a) Training sub-image



b) Hand drawn smoothed target image

Figure 8.17: Training sub-image and target

An intensity profile taken along a horizontal line about two thirds of the way down the training sub-image(marked by a white horizontal line) is shown below in Figure 8.18. While the main trunk edge positions can be seen within this

profile it is clear that there are many other similar edges within the image and that the task is not a trivial one.

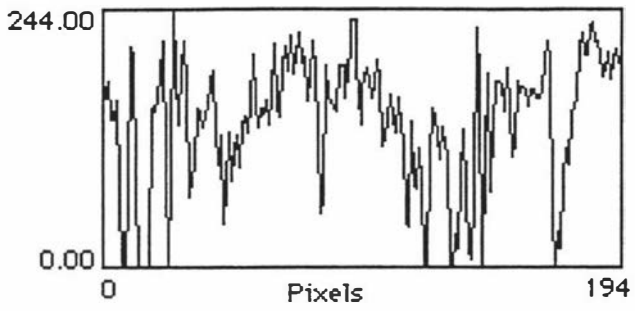


Figure 8.18: Intensity profile through training sub-image

a) Section of output from NNWF

b) Section of output from NNWF + Original image

Figure 8.19: NNWF Result from standard training

8.2.2.1 Consideration of S-R Characteristics

To avoid possible problems with the unnatural precision produced by spatial quantisation smoothed targets were used for all the experiments. A single experiment was performed to compare the results of training with sharp edge targets and training with smoothed targets. This was done using the reference training method M0. The results appeared subjectively similar but as the target has been changed direct comparison of the RMS errors is not appropriate.

In considering the stimulus response characteristics and the results from the simple training experiment shown above one difficulty appears to be that left hand edges are different from right hand edges but the target is the same. In fact left hand edges, while admittedly less pronounced, seem to have been entirely ignored by the NNWF. One aim of considering S-R characteristics of the task is to make those input vectors which are to be discriminated as different as possible and those vectors which are to be generalised as similar as possible. To make discrimination easier the task can be separated into two tasks; one is to find the left hand edges the other is to find the right hand edges. The two outputs can then be combined. The results of using this approach are shown below in figure Figure 8.20. A marked improvement is apparent with considerably more of the left hand edge of the central tree visible.

a) Result of training on left hand side

b) result of training on right hand side

c) Combined result (a+b)*0.5

Figure 8.20: Separate left right training

8.2.2.2 Sub-tasks

The use of sub-tasks to break the main task down into manageable steps is discussed in detail in the previous chapter. The main difficulty in applying this heuristic is the identification of suitable sub-tasks. Several sub-tasks were identified for possible use in the detection of tree edges and are discussed below.

The first feature evaluated as a potentially useful sub-task was the detection of areas within the image containing the distinctive texture of sharply focused tree bark. These areas were marked by hand and then a network was trained to fill in these sections of the image. This trained network was then combined with the original network using methods M2, M3 and M4 described previously. The most successful of these was the pre-training method M2. The output image formed by a network trained to perform this sub-task (Figure 8.21) and then trained to perform the main task is shown in Figure 8.22. Note this network is only being trained to find left hand edges so comparison should be with Figure 8.20. Subjectively this only appears slightly better and the comparison with target produced similar RMS errors.

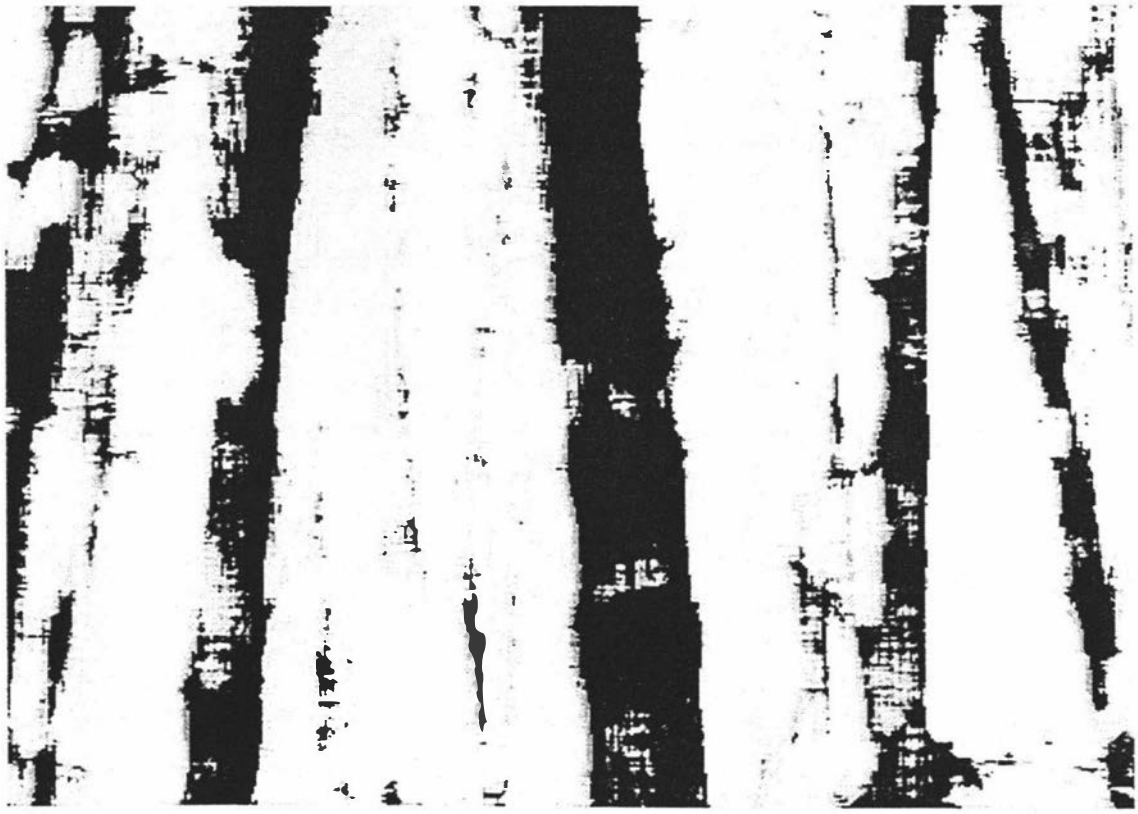
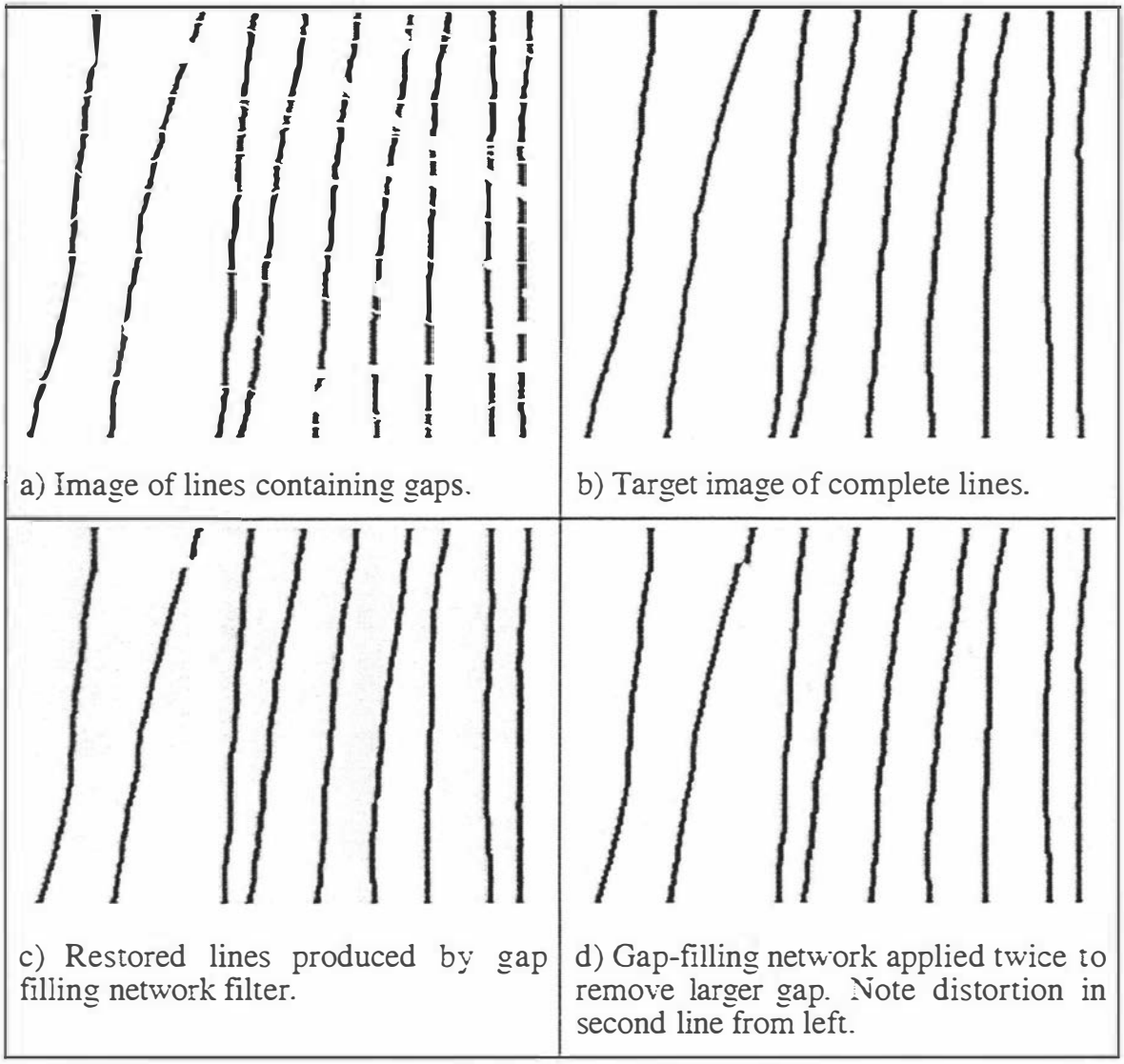


Figure 8.21: Output from "find bark" sub-network

Figure 8.22: Output from network pre-trained on "find bark"

As a second sub-task; the forming of continuous lines was evaluated. If the final task is to draw a single continuous line down each side of each tree in the foreground then the removal of small gaps in these lines is possibly a useful sub-task. This is a sub-task towards the right hand end of a solution graph. The final tree edges should be continuous so the network should "fill-in" gaps in the tree edges found by previous stages of processing. To train a network to perform this task a series of roughly vertical lines were hand drawn and smoothed. Gaps were then created by rubbing out small sections of these lines. A network was then trained to restore these gaps. Finally this trained network was used to post-process the output from the main network. The results are shown below in Figure 8.23 and Figure 8.24. The extent to which gaps can be filled will be limited by the size of the input window, unless the network is applied iteratively. Extended use of such gap-filling may also introduce false edges when either large gaps are filled or gaps in edges at angles which were not adequately represented in the training set are filled.



a) Image of lines containing gaps.

b) Target image of complete lines.

c) Restored lines produced by gap filling network filter.

d) Gap-filling network applied twice to remove larger gap. Note distortion in second line from left.

Figure 8.23: A gap-filling network

A section of the output image produced by the NNWF when trained to find left hand edges and then post processed by the trained "gap-filling" network is shown in Figure 8.24.

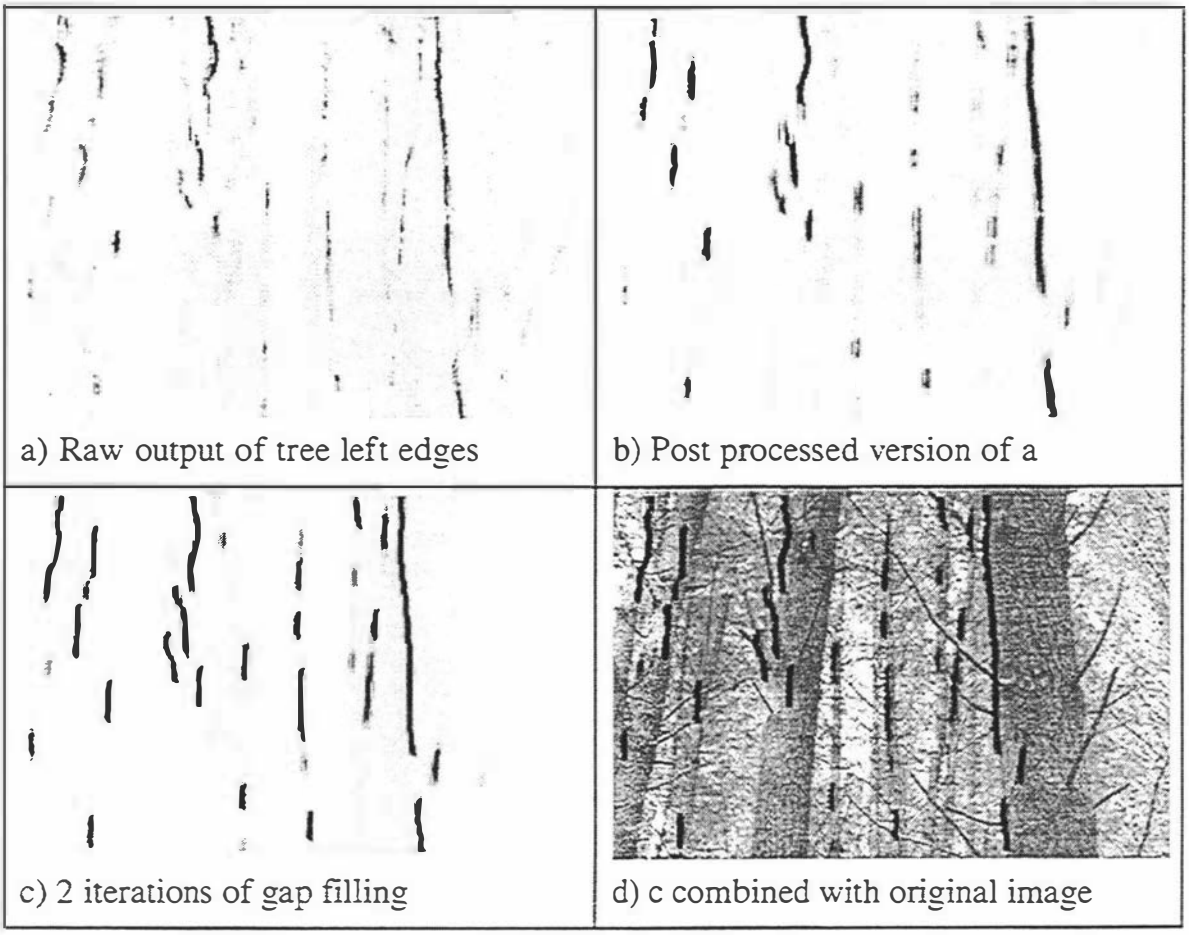
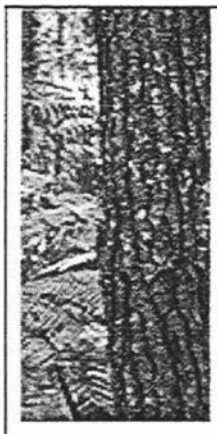


Figure 8.24: Post processing with gap-filling network

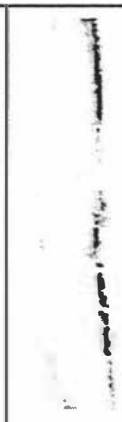
8.2.2.3 Simpler-Similar Tasks

As a simpler similar task the network was trained on a small consistent section of a tree as shown in Figure 8.25a below. This pre-trained network was then trained on the main task with the results shown below. This did not reduce the training time on the main task or improve performance on the main task.

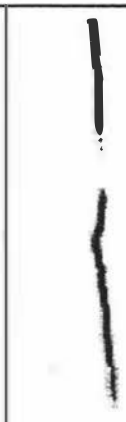


a) Simpler-similar task input

b) Simpler-similar target



c) Trained net output



d) c processed by gap-filling network

d) Performance on "find left hand edges" task after pre-training on a simpler-similar task .

Figure 8.25: A simpler similar task

8.2.2.4 Use of Rules

The rule that the filter should be symmetrical can be used to provide extra training from the same sub-image. As a separate network is being used to find left and right hand edges extra training for the left hand network can be produced in the following way. First a network was trained on left hand edges as above. Then the training was continued but using a horizontally flipped version of the input sub-image and a similarly transposed version of the training target for finding right hand edges. The results of using this augmented training set are shown below in Figure 8.26.

Figure 8.26: Training set extended by use of a rule

8.2.2.5 Structured Training

Several types of structured training were investigated in attempts to improve the network performance. These are described in two groups below:

A1: First the training procedure was modified to use only those portions of the original image which corresponded to dark areas in a gating image. This gating image was produced by combining the original smoothed target with the output image produced by a partially trained network. This image was then thresholded to provide a binary gate signal. This image was then blurred and re-thresholded to include a greater area for training. The output from an network trained in this way is shown in Figure 8.27. In this case structuring of the training did not provide any improvement in performance. Again only the finding of left hand edges is being tested.

a) Net output after training on selected parts of image.

b) Net output after further training on main task.

Figure 8.27: Structured training 1

A2: A second experiment was performed using a selection of special training areas around edges which were not being identified by the NNWF. This was used to gate additional training. This also did not provide any improvement in performance.

A second method of using structured training is to combine the other heuristics. Several combinations of the results from heuristics one to four above were tested:

B1: Two networks were used N1 and N2. N1 was pre-trained on the texture sub-task and then trained to find smoothed left hand edges. N2 was trained as a "gap-filling" network. The two trained networks were then cascaded in a sequence (method M1). This trained sequence was first used to detect left hand edges and then re-used on a reversed input image to find right hand edges. The two resulting output images were then added together. The results are shown below in Figure 8.28.

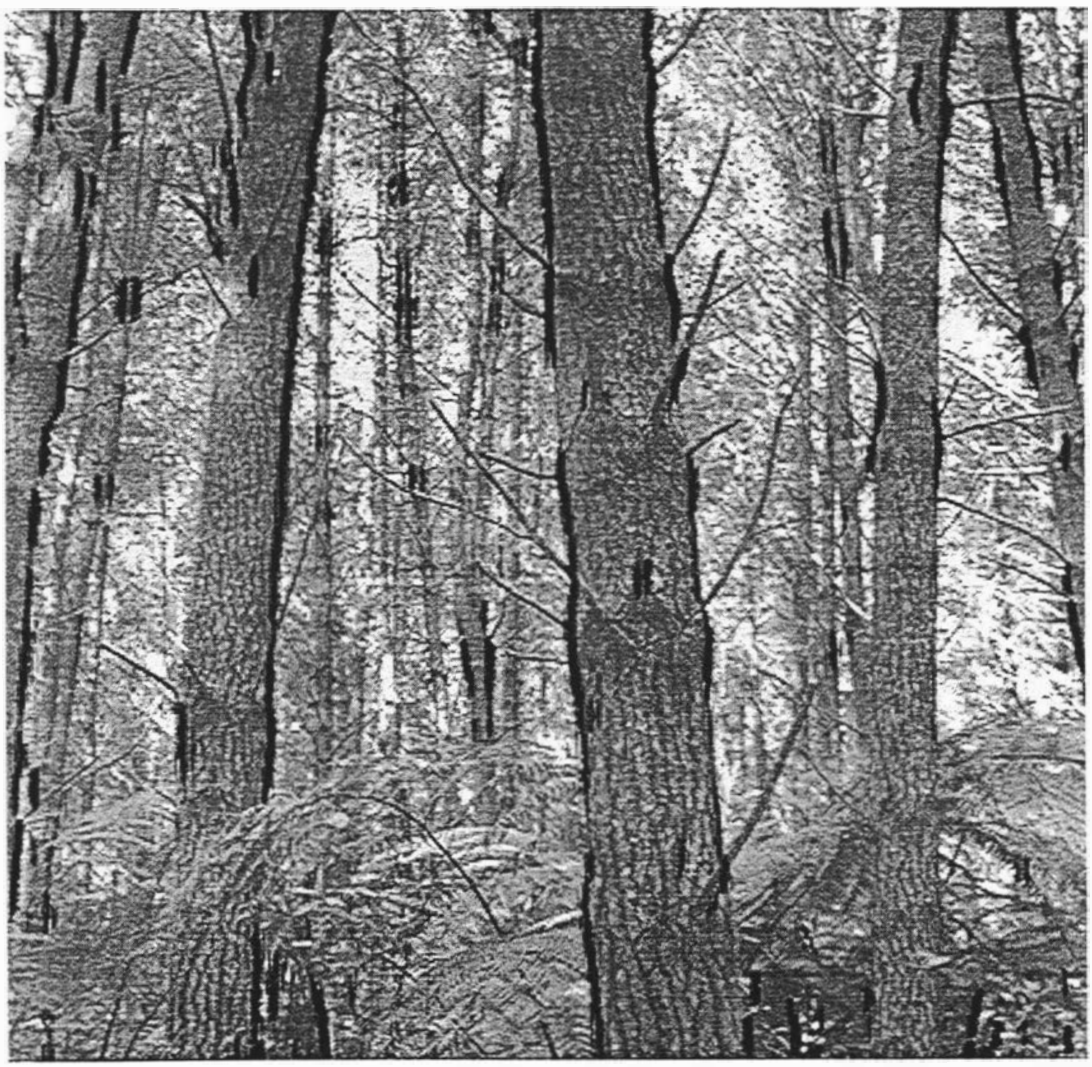


Figure 8.28: Combined result

The same trained networks was then used to process another tree image, taken on a different day in a different area of forest. The result is shown in the two images below:

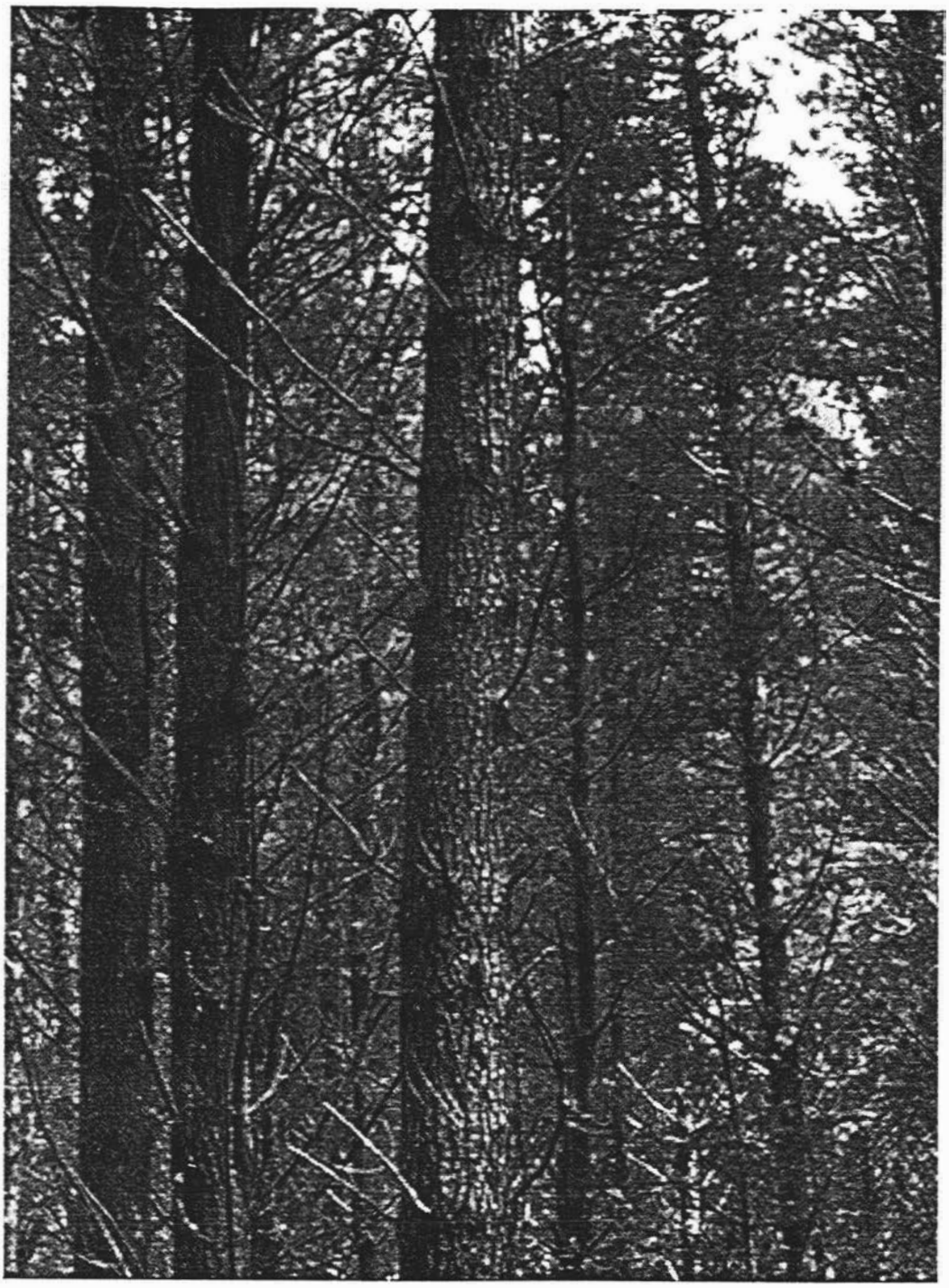


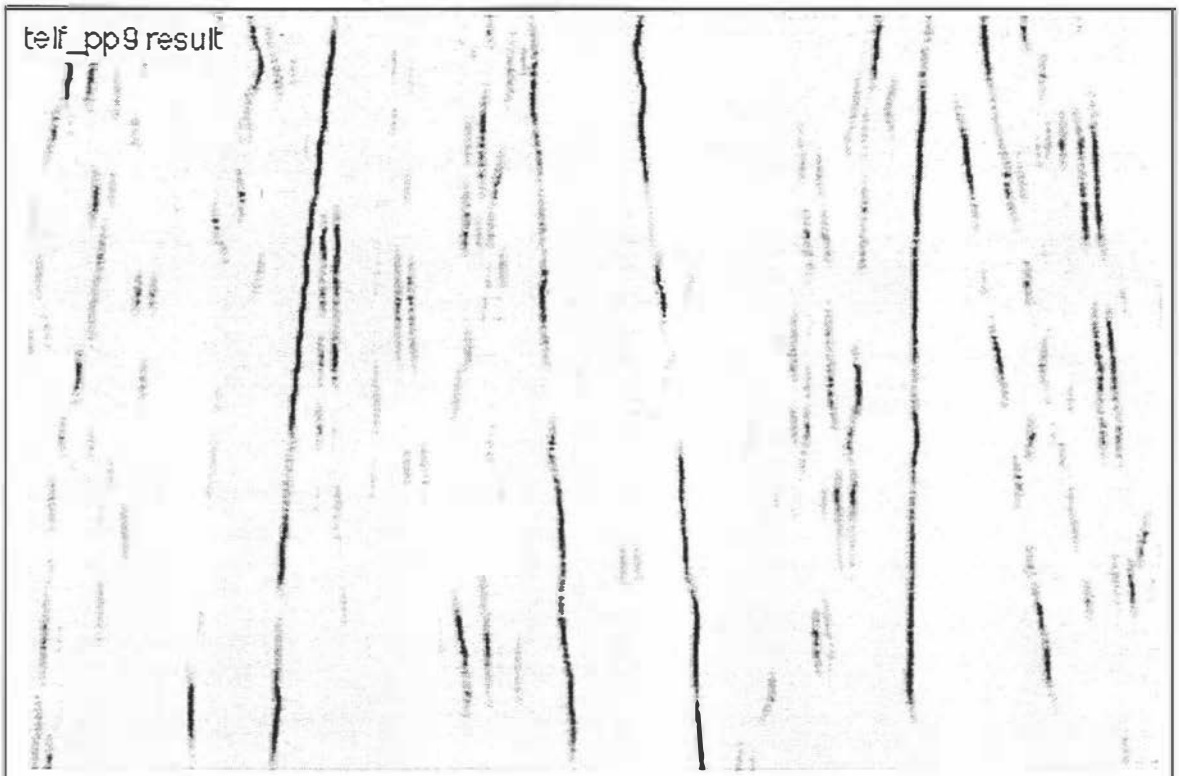
Figure 8.29: A different forest scene

Figure 8.30: Edges found by NNWFs

B2: A similar but extended approach was to pre-train the second network N2 as a gap-filling network but then to continue training it with the image produced by N1 as an input and the original smoothed left hand edges as a target. This is using the gap-filling task as a simpler similar task for pre-training a post processing network. The results for this approach are shown below, again using the trained networks twice; once for left hand edges and once for right hand edges.

Figure 8.31: Results from post processing network

B3: Finally the same procedure was used but with the input for N2 provided by a network which used the rule extended training of 8.2.2.4. The results of this experiment are shown in Figure 8.32. Both the output combined with the original image(b) and the raw NNWF output(a) are shown.



a) Raw output of combined NNWFs

b) Output combined with original input image to show position of found edges

Figure 8.32: Post processing of rule extended result

8.2.2.6 Discussion of Tree edge experiments

The finding of tree edges was used as an example because it is a **real** rather than an **artificial** task. It is important to consider such real tasks as well as more controlled and manufactured tasks if the results of these investigations are to be of general use. However, real tasks also bring difficulties. We cannot directly compare the results of all the experiments tried as there is no single perfect answer with which the NNWF output can be compared. The small window size may also have affected the final performance level but was required to provide reasonable simulation times.

It is also important to note that the application of the suggested heuristics is not like testing the use of a formula for calculating the volume of a sphere. Learning a sub-task first will only help if the right sub-task is chosen and structuring the order of presentation of examples will only help if a useful ordering can be found. For these reasons while some of the experiments above did not produce improvements this may only indicate that the particular examples were not well chosen. The heuristics themselves may still be very appropriate.

A second difficulty with this example is the comparison of performance. Because no definite **correct** target exists comparisons are subjective. Only a few clear edges of the main trees were produced by hand. A network which finds other tree edges in other parts of the same image or in other images may be judged to be doing better or worse depending on the requirements of the experimenter. Table 8-5 provides a comparison of the results from the experiments above. Two methods of comparing performance were used:

- Firstly, a subjective evaluation of the image produced by the trained NNWF was made. In each case it was compared with the image produced by standard training. This included a subjective evaluation of how difficult it would be to produce an algorithm to link the found edges together to form shape description of the trees in the foreground and to ignore the other extraneous background edges.

- Secondly, an "objective" measure was made. The RMS difference between the image produced by the NNWF and a smoothed target image containing hand drawn edges of the four trees in the forground was calculated. This RMS error was then compared with that for an image produced by standard training.

In general there is a good correlation between the objective and subjective measures. Where they differ it appears to be due to the difference between a loosely defined subjective goal and the more specific but perhaps less "appropriate" goal provided by the RMS difference measure. For example, when separate left and right training was used the left edges of the tree were found. These were almost entirely ignored by standard training (compare figure 8.19a and 8.20c), however the RMS difference from target **is** actually higher. A filter which ignored left hand edges, while presumably producing more accurate right hand edges, was not what was required but it was implied by the RMS difference measure.

Figure	RMS Error	Objective assessment	Subjective assessment	Description
	40.4	**standard**		standard training
8.19	39.9	better	better	smoothed target
8.20c	41.6	worse	better	separate L-R edges
8.22	39.2	better	better	bark sub-task
8.25	45.1	worse	similar	simpler similar task
8.26	40.0	better	better	rule extended training
8.27a	43.3	worse	similar	structured training A1
8.27b	45.2	worse	similar	structured training A1
8.28	45.9	worse	better	structured training B1
8.31	38.75	better	better	structured training B2
8.32	36.2	best	best	structured training B3

Table 8-5: Comparative results

Depending on the intended use of the output images the indication of other tree edges may be an advantage, of no consequence, or a disadvantage. One possibility would be to take the network output, remove unwanted edges by hand and then to use this modified image as a new target for further training. Such a form of successive refinement of the goal could be compared to the iterative knowledge extraction procedure used with some expert systems. By using the expert system to produce an output which is then evaluated by the expert the expert knowledge is gradually extracted.

8.3 Discussion

The two sets of experiments reported in this chapter involved a considerable amount of computer simulation and analysis time. However, they represent only a very small sample of possible tests of the heuristics suggested for providing a beneficial educational environment. The first task demonstrated the use of a number of methods of implementing a single heuristic. It was also a relatively contrived task which provided clear objective measures of performance. The second series of experiments involved the application of each of the heuristics to a single real world task. By using these two different sets of experiments a broad picture of the advantages and difficulties involved in using an educational environment to improve neural network performance has been demonstrated. The results show that improvements in performance can be achieved, that a number of different implementation methods can be used and that different methods may be appropriate for particular tasks. The difficulty of choosing suitable sub-tasks, simpler-similar tasks, rules and targets has also been highlighted.

In the introduction to this chapter it is pointed out that the aim of the chapter is to validate the general approach rather than to suggest or verify a complete theory of instructional design for ANNs. This more limited aim as been achieved but the experiments have also shown that there is scope for further experimentation and refinement of the approach.

Chapter 9

9. SUMMARY & CONCLUSIONS

This thesis describes research which began with the general aim of investigating the use of neural network techniques in image processing. This was refined to a more specific aim of constructing and experimenting with a neural network based window filter. The subsequent construction and testing of such a filter led to an appreciation of the intricacies and difficulties involved in implementing and training artificial neural networks. A NNWF was implemented and trained to perform a number of standard filtering operations. Following the difficulty found in teaching the NNWF to perform certain more complex tasks a number of existing performance improvement techniques were examined, some of these were incorporated into the NNWF. Consideration was then given to the question; How do intelligent systems arise and what are their general characteristics? From this speculative analysis came an appreciation of the possible importance of a training or educational environment. An investigation of possible features of such an environment for the NNWF and a review of the foundations of instructional design for people was then carried out. The formation of a set of instructional design heuristics for neural networks, an investigation of methods of applying these heuristics to the NNWF and some initial demonstrations of their effective use forms the last part of the research described in this thesis.

9.1 SUMMARY

The three key fields: window filters, neural networks and instructional design have been introduced and the reasons for combining these areas have been explained. A neural network based window filter has been designed and implemented as described in chapter three.

It has been demonstrated that general filter operations can be defined by sample input-target image pairs and that the NNWF can be trained to perform these filtering operations. It has also been shown that the NNWF can be trained to perform some tasks which are difficult to perform by other means.

The difficulty in training the NNWF for certain more complex tasks has been indicated and standard methods of alleviating this by improving the "pupil" have been briefly reviewed.

Some speculations on the nature and evolution of intelligent systems has been presented. Following this discussion and consideration of the possible functions of logical sequential thought (LST) in intelligent systems an alternative role for LST has been suggested. This use of LST for the transfer of knowledge suggests the possible importance of a training or educational environment for intelligent systems.

In an attempt to develop an educational environment for the NNWF a set of instructional design heuristics for artificial neural networks has been proposed. Using the analogy of human instructional design a more formal basis for an extended set of heuristics has been developed.

A number of methods of implementing these heuristics for the NNWF have been suggested and many of them investigated experimentally.

Finally in order to demonstrate the use of these instructional design heuristics some initial experiments are described in which improvements in performance of the NNWF are brought about by the use of the proposed heuristics to create an educational environment for the NNWF.

The limitations of any window filter, which can only make use of local information is apparent in some of the results. It was necessary to constrain this research to the use of filters with a relatively small input window size in order to

enable experiments to be performed in reasonable time frames on the computers available. However, hardware implementation of the NNWF and the use of pyramidal or multi-resolution input windows may remove this restriction. It was always intended that the NNWF would be used as a local operator, however the potential for augmenting the NNWF with non-window inputs also exists and warrants further study.

9.2 CONCLUSIONS

The combination of window filters and neural networks has been shown to be a useful approach for investigating the use of neural networks in image processing. This combination has allowed a number of aspects of neural network training and implementation to be investigated using simulations which could be performed in a reasonable time. Despite this, the time required to train network simulations has still been long enough to limit the number of avenues which could be investigated. The inherent limitations of window filters have also at times made the identification of suitable experimental tasks difficult.

The universal neural network based window filter has been shown to be a useful tool for image processing. It has been shown that it can learn general operations from input-target image examples. It has also been demonstrated that it can be trained to perform tasks which are difficult to perform by other means. This suggests that there is potential for a high speed hardware implementation of the NNWF. Such a system would be useful even if only a high speed run mode was provided, together with a slower software based training mode.

A number of possible methods for the creation of targets for training the NNWF have been suggested: Target creation by a domain expert, target creation by an image processing expert, target creation by special one off procedures (for example aggregation of multiple exposures to reduce noise), and target creation by reversing the normal flow of operations (for example adding noise or image imperfections and using the original image as the target). Only some of these methods have been used in the experiments presented and there are interesting research possibilities for testing and extending this list of techniques.

The use of a training or educational environment for ANNs has been shown to be beneficial and to be an area which warrants further study. A number of methods of incorporating the suggested instructional design heuristics into a NNWF have been demonstrated and a start has been made on the construction of a theory of instructional design for artificial neural networks.

There is considerable scope for further research on:

- The hardware implementation of small neural network based window filters and the methods of incorporating these into general purpose image processing systems.

- The methods of providing training targets for the NNWF.
- The use of larger or multi-resolution windows, non-square windows, pyramidal neural network based window filters and network window filters augmented with non-local inputs.
- The design of experiments to test the validity of some of the other speculative ideas on aspects of intelligence and the avoidance of the frame problem suggested in chapter four.
- The methods of implementing the suggested instructional design heuristics, particularly the use of rules in training and the incorporation of algorithms and hints, into neural networks.
- The application of the educational environment techniques to other types of network architecture and other neural network application domains such as control.
- The use of educational environments for artificial neural networks as a test-bed for evaluating theories of instructional design for people.

APPENDIX A

MARR-HILDRETH OPERATOR EXAMPLES

This appendix provides some further examples of the use and properties of the Marr-Hildreth operator. The Marr-Hildreth operator used in the research described in this thesis was implemented as an NIH Image plugin filter. The theory of operation is described in chapter two [§2.1.3] and the source code is provided in part one of appendix C. Part of the description of the filter contained in chapter two is repeated here to introduce the examples.

Marr & Hildreth suggested that the purpose of early visual processing is to construct a primitive but rich description of a visual scene which can be used to determine the reflectance and illumination of visible surfaces, and their orientation and distance relative to the viewer. On this basis they developed an edge-detection filter which extracts zero-crossings after convolving the image with a Laplacian of Gaussian mask. This has become known as the Marr-Hildreth operator or edge filter [Marr & Hildreth].

The MH operator used for this work used the following implementation. A mask is calculated using the formula $\nabla^2 G(x,y) = \frac{1}{\pi\sigma^4}\left(1 - \frac{(x^2+y^2)}{2\sigma^2}\right)e^{-(x^2+y^2)/2\sigma^2}$ which is then convolved with the input image to produce a real valued image. The value of sigma in this equation effectively determines the width of the ideal mask.

Spurious zero crossings were removed by setting to zero any resultant pixels with a magnitude less than a threshold. The range of values from zero–threshold to zero+threshold is referred to as the deadband. Zero crossing points are then found by examining opposite pairs of the eight surrounding pixels at each position. If a sign change is found then an edge is considered to be present. Two output alternatives are provided;

1. If a zero crossing is found an output is produced in which the rate of change at the zero crossing is represented as a grey scale.

2. Alternatively, the inverse of the distance from the actual zero crossing is calculated and this is represented as a grey scale.

At each position the result of the convolution for that pixel(c) and the eight surrounding pixels is considered. The four pairs of pixels on either side of c are examined to look for a change of sign. If a change of sign is found then a value s is calculated which is the largest difference between pixels on either side of c. The output value z is then calculated as

$z = s - c$ for scale by rate of zero crossing

$z = 1 - \frac{c}{s}$ for scale by distance from zero crossing point

This results in pixels which occur exactly on a zero crossing being shown as black. Where the zero crossing occurs between two pixels both pixels have a

grey level representing their relative distance from the zero crossing. This provides a grey level image which is a form of anti-aliased edge strength map to sub-pixel resolution. For both output forms the resultant edge image is scaled to use the full range of the 8 bit grey scale representation. Another common approach is to provide an output image of greater resolution than the input image to enable the edge position to be indicated to sub pixel accuracy. This approach was not suitable for this work as an output image of the same size as the input image was required.

A question which arises in the implementation of the MH operator is what to do about truncation of the mask due to the square shape of the convolution window? This is particularly important when small window sizes are used for large values of sigma. Two approaches were used here;

1. The values within the mask should add to zero if no truncation has taken place. Therefore once a sigma value has been chosen a window size can be calculated to provide a mask truncation error of less than a set value. The value used in this implementation was 0.01.

2. The second approach is to add an offset to all the values within the mask to make it sum to zero.

Marr also suggested that the image should be convolved with a number of masks each using a different value of sigma. The idea was that an edge representing a real object should appear throughout a range of spatial frequencies. This post processing stage which combines the output of a set of MH filters is not used in the work described in this thesis.

Figure 1 below shows the effect of using the MH filter on a variety of simple objects and shaded regions. The parameter values used for each sub-image are shown in the image: s is the value of sigma, w is the window size, zb is the zero band for which zero crossings are ignored, o=dist implies the output is coded based on distance from zero crossing and o=magn implies that the output is coded based on the rate of change at the zero crossing. The input image is shown in figure 1c. The effects of using the two different output coding schemes are shown in a and b. Some of the phantom edges produced by second derivative

operators at T intersections  and by objects in close proximity  can be seen in b. The effect of truncation when a dead band is not used is seen in d.

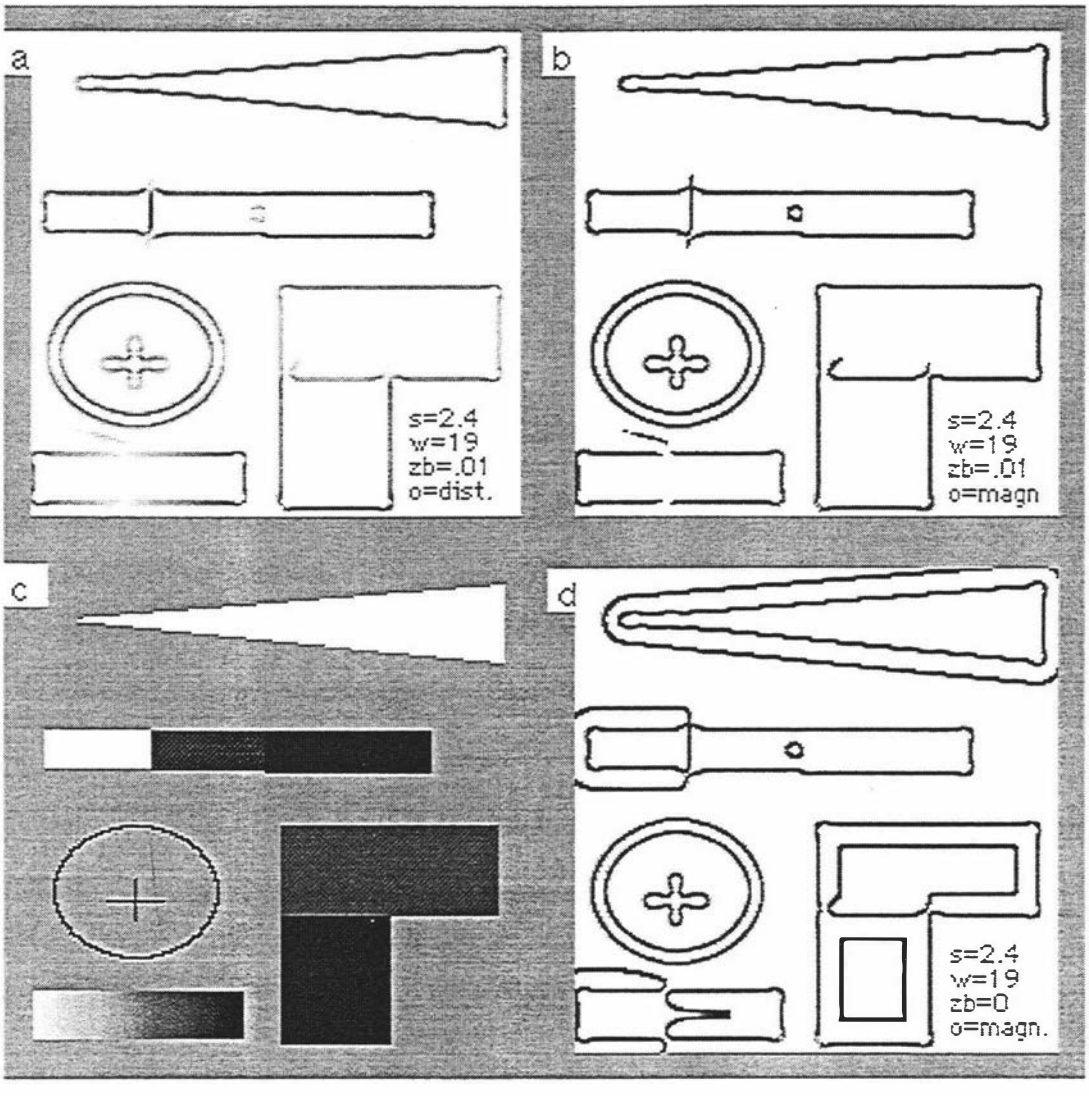


Figure 1

In Figure 2 the effects of increasing the value of sigma is shown. Note the distortions, phantom edges and undetected edges. Automatic selection of window size, scale output by magnitude and a deadband of 0.01 were used for all these sub-images. As sigma is increased corners become rounded and the fine edges on the greyscale wedge (visible with sigma=0.8) are lost.

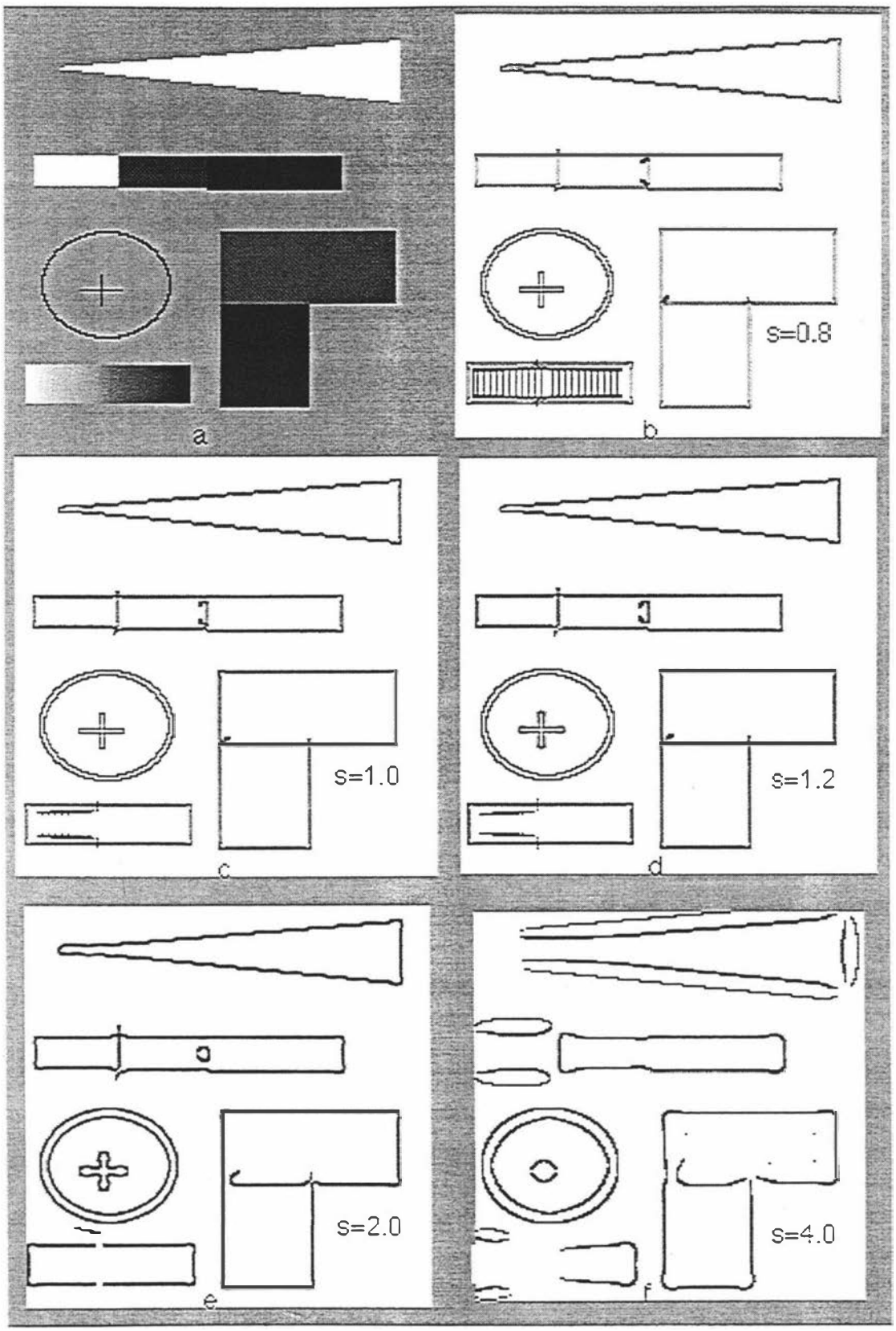
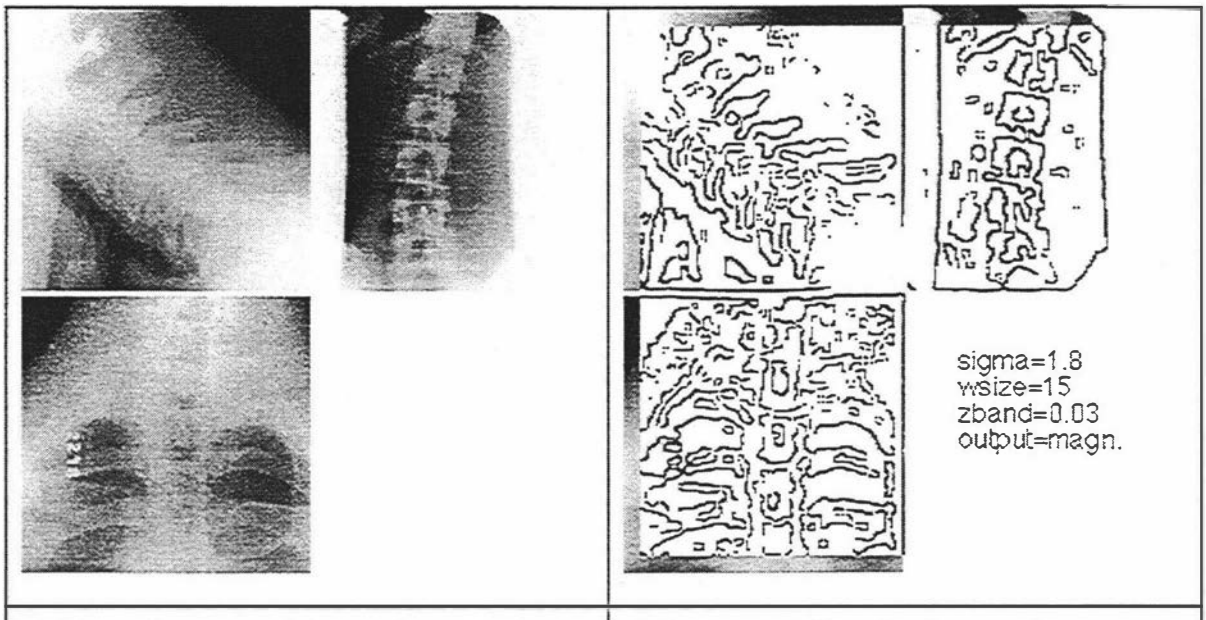


Figure 2

Figure 3 shows the use of the filter to detect the edges of bones in x-ray images. The parameter values used are shown within the image. By careful selection of parameter values it was possible to provide good edge information in these rather difficult images.



a) X ray images of the neck and spine

b) Output of filter for images in a

Figure 3

In Figure 4 the result of using the filter to find the edges of selection of simple objects on a table top is demonstrated. The filter output with sigma values of 1.2, 2.4 and 3.6 are shown. The auto window size, scale by magnitude and a zeroband of 0.01 was used for all of these. Larger values of sigma reduce the background clutter but also distort the edge information for the main objects.

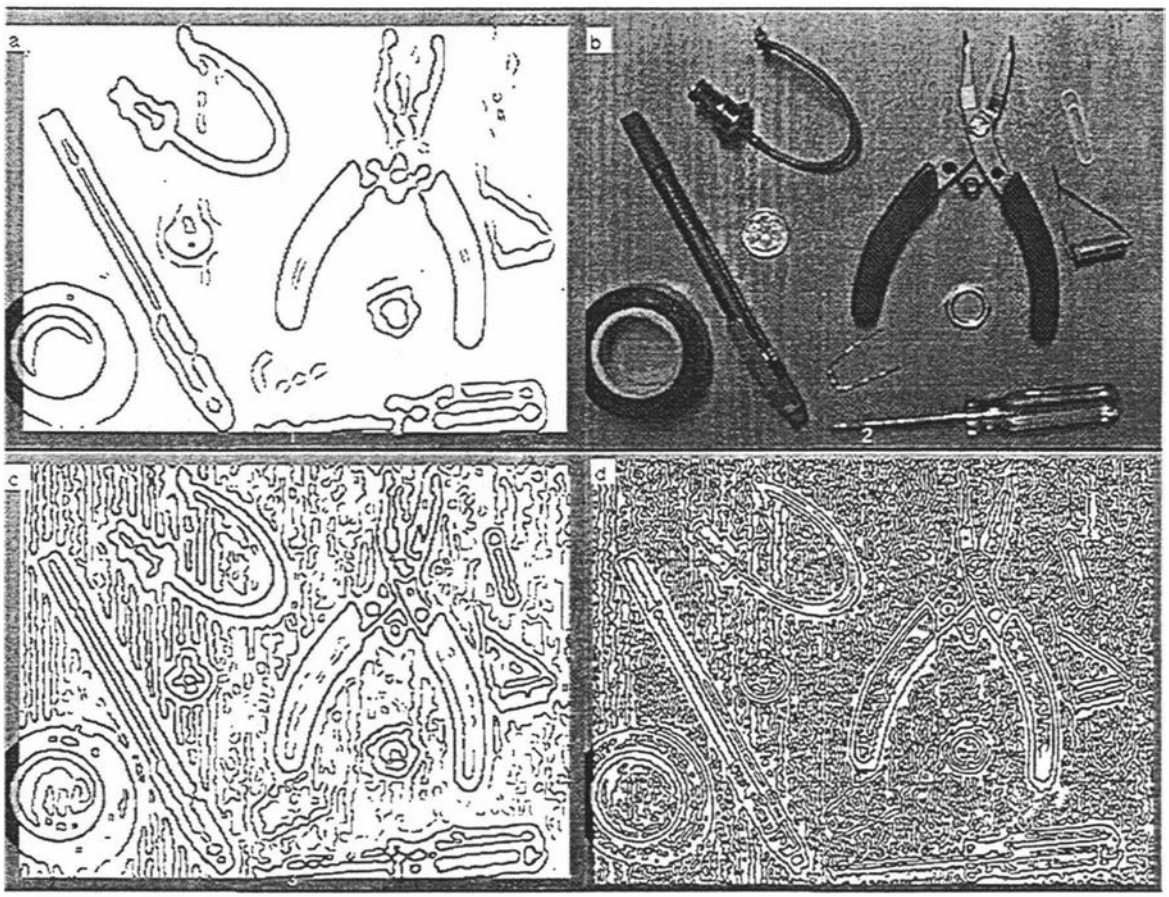


Figure 4

a) sigma = 3.6 c) sigma = 2.4 d) sigma = 1.2

In Figure 5 the original image was first expanded using a times two bilinear scaling. The edges were then detected using the Marhil operator with a sigma value of 3.6 and then the image was scaled back to original size. This is equivalent to using a sigma value of 1.8 in terms of frequency content of detected edges but is less affected by the noise in the original image as can be seen by comparing this with Figure 4c and 4d.

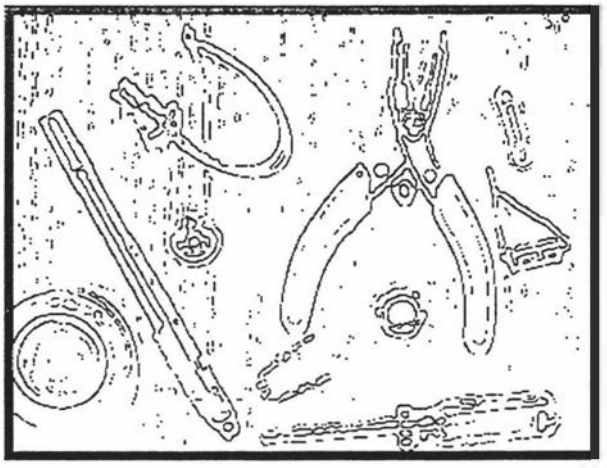


Figure 5

The Marr-Hildreth operator was used in this thesis as an example of a well known and biologically based filter which the NNWF could be trained to emulate. However, the implementation details are rarely discussed in the literature and some of the subtle effects shown above only became clear once the plugin filter had been produced and tested.

APPENDIX B

ASPIRIN/MIGRAINES ADDITIONS AND UNIX CONTROL FILES

To implement a NNWF the Aspirin/MIGRAINES package had to be modified and a system of control files designed. This appendix contains the source listings for the modifications made to the Aspirin/MIGRAINES package and examples of the Unix control files used in the NNWF experiments. The listings are:

- User_Init.c
- Learn control file
- Run control file
- Dump weights control file

INITIALISATION FILE

The main modifications are included in the user_init file shown below, in addition it was necessary to make some changes to the backprop.c file to make use of calculated epoch sizes and to allow the use of verification images in addition to learning images. The user_init file performs the following actions:

- Initial loading into memory of sub-images from input, target and verify images.
- extraction of input window and target value at each position

```
// The user_init function is at the end of this file and is called by AM at initialisation
// time. The various functions that it calls are at the beginning of this file.

#include "sdn.h"
#include  "../sources/rhpclib.h"
//rhpclib contains definitions of utility routines which allocate, initialise  and release
//memory used for two dimensional dynamic arrays


#define MAX_TSETS 5          // maximum number of input-target image pairs
#define WSIZE 9              // default window size
#define TLX 0
#define TLY 0
#define XSIZE 200            // default sub-image size
#define YSIZE 100
#define  INPUT_FILE "../images/se1.mcid"        // default input and target image files
#define TARGET_FILE "../images/se1mh.mcid"

//Global declarations - these variables are used within backprop.c
int Target_set=0;        /* flag to say if testing or learning used as an index into i/o
                         images */
int iosets;              /* number of image pairs */
int wsize;               /* window size (which enables epoch size to be calculated)*/
int xsize[MAX_TSETS],ysize[MAX_TSETS]; /* gives epoch size for each data set */
int repoch[MAX_TSETS];                 /* gives epoch size for each data set */

// Static globals ie only visible to routines in this file
static int tlx[MAX_TSETS],tly[MAX_TSETS];
static float input_window[1683];              /* implies max window size is 41x41*/
static float target_pixel[1];
static char input_file[100];
static char target_file[100];
static int wx[MAX_TSETS],wy[MAX_TSETS];   /*  latest position of window in inp image
static int tx[MAX_TSETS],ty[MAX_TSETS];      /*    latest pos. of target pixel in target
// declare descriptors for the input and target image structures
static imgd target_image[MAX_TSETS],input_image[MAX_TSETS];

//-------------------------------
// Routine to open the input and target images
```

```
static void open_images(int l){
   FILE              *inf;
   int               fx,fy;
   unsigned char     lobyte,hibyte;
   int                x,y;

   printf("\nOpening input image %s...",input_file);
    if ((inf = fopen(input_file,"r"))==NULL) {
        printf("Error opening %s\n",input_file); exit(0);
   }
   // Read image dimensions from file File is in .MCID format see NIH Image for details
   fread(&lobyte,1,1,inf); fread(&hibyte,1,1,inf); fx = lobyte + hibyte * 256;
   fread(&lobyte,1,1,inf); fread(&hibyte,1,1,inf); fy = lobyte + hibyte * 256;
   printf("x,y=%d,%d\n",++fx,++fy);        //Display image dimensions from file
   dimimg(&input_image[l],fx,fy);          //Get space for image and fill in descriptor
   // Read in the image
   for (y=0; y<fy; y++){
       if(fread(&input_image[l].e[y][0],1,fx,inf) != fx){
              printf("Error while reading input file\n"); exit(0);
       }
   }
   fclose(inf);
   printf("\nOpening target image %s...",target_file);
   if ((inf = fopen(target_file,"r"))==NULL) {
       printf("Error opening %s\n",target_file); exit(0);
   }
   /* Read image dimensions */
   fread(&lobyte,1,1,inf); fread(&hibyte,1,1,inf); fx = lobyte + hibyte * 256;
   fread(&lobyte,1,1,inf); fread(&hibyte,1,1,inf); fy = lobyte + hibyte * 256;
   printf("x,y=%d,%d\n",++fx,++fy);
   dimimg(&target_image[l],fx,fy);          // get space for image and fill in descriptor
   // read in the image
   for (y=0; y<fy; y++){
       if(fread(&target_image[l].e[y][0],1,fx,inf) != fx){
              printf("Error while reading target file\n"); exit(0);
       }
   }
   fclose(inf);
}
```

```
//-------------------------------------------------------------
// Routine to get the next input window from the input image and transfer it to AM
static void gen_iwindow(){
   int  x,y,
        l,      // index into array of training sets
        i=0;

   l = min(Target_set,(iosets-1));
   for(y=wy[l]; y<(wy[l]+wsize); y++){
        for(x=wx[l]; x<(wx[l]+wsize); x++){
              input_window[i++] = input_image[l].e[y][x] / 255.0;
        }
   }
   if(++wx[l] > (xsize[l] + tlx[l]  - wsize)){
        wx[l] = tlx[l]; wy[l]++;
        if(wy[l] > (ysize[l] + tly[l] - wsize)){
              wy[l] = tly[l];
        }
   }
   sdn_set_input(input_window);     // transfer it to aspirin/migraines
}

//---------------------------------
// get the next target pixel from the target image
static void gen_tpixel(){
   int l;
   l = min(Target_set,(iosets-1));
   target_pixel[0] = target_image[l].e[ty[l]][tx[l]++] / 255.0;
   sdn_set_target_output(target_pixel);    /* transfer it to aspirin/migraines */
   if(tx[l] >= (xsize[l] + tlx[l] - wsize/2)){
        tx[l] = tlx[l] + wsize/2; ty[l]++;
        if(ty[l] >= (ysize[l] + tly[l] - wsize/2)){
              ty[l] = tly[l] + wsize/2;
        }
   }
}

//-------------------------------------------
//Routine called by AM each time a new training pair is required
static void set_pattern(){
   gen_iwindow();
   gen_tpixel();
}
```

```
/*****************************************************************/
/* This function is called at beginning of simulation            */
/* It must initialise - ie read in input and target images       */
/* Tell am what functions to call to when it next input/target pattern pair */
/* These functions must in turn get the input and target values and call the */
/* set_inputs and set_target am functions                        */
/* and pass them pointers to the input and target arrays         */
/*****************************************************************/
user_init(){
   extern void define_generator();  /* from Backprop.c */
   int I;
   get2i("\nEnter window size, number of training sets",9,2,&wsize,&iosets);
   printf("\nWindow size is %d. ",wsize);
   printf("Number of input/output image sets is %d\n",iosets);
   for(I=0; I<iosets; I++){
       strncpy(input_file,sget("\nEnter input image filename",INPUT_FILE),80);
       strncpy(target_file,sget("\nEnter output image filename",TARGET_FILE),80);
       get2i("\nEnter training subimage size",128,128,&xsize[I],&ysize[I]);
       get2i("\nEnter subimage offset",0,0,&tlx[I],&tly[I]);
       repoch[I] = geti("\nEnter epoch size",(xsize[I]-wsize+1)*(ysize[I]-
                    wsize+1));
       printf("\nTraining subimage is %d by %d at
              %d,%d\n",xsize[I],ysize[I],tlx[I],tly[I]);

       printf("Epoch size is %d\n",repoch[I]);
       printf("Training subimage is %d by %d at
              %d,%d\n",xsize[I],ysize[I],tlx[I],tly[I]);
       wx[I]=tlx[I];   wy[I]=tly[I];
       tx[I] = tlx[I] + wsize/2; ty[I] = tly[I] + wsize/2;
       open_images(I);
   }
   Target_set = 0;
   define_generator(set_pattern, (char *)NULL, "Gen input and target");
}/* end initialise */
```

LEARN CONTROL FILE

The following is the contents of a typical learn control file for use with the modified aspirin/migraines package. It defines the input, target and verify image files, sub-image sizes, window sizes, network training parameters and the log file for results. It then executes the network simulation produced by AM and logs the results to disk. Explanations are given in italics.

```
#!/bin/sh
set sname=tshapes      base file name for input and target images
set fname=ts1          experiment, used for log file and save file for network weights
set x=506              sub-image dimensions
set y=104
set w=9                window size
set epoch=`mult $x $y $w`          calculate epoch size using utility program mult
the next line executes a program generated by aspirin migraines and the user_init file
above. w9h5k3d indicates it was for a 9x9 window with two hidden layers with 5 and 3
neurons respectively and one output neuron. The use of the various switches for the
command line are explained in the aspirin/migraines user manual
/bin/time ../w/w9h5k3d  -F $fname -l -s ${epoch} -t ${epoch} $epoch .001 -a .01 -
i 0.9 -m 0.01  -l ${epoch}00 << end >! $fname.log &
The following lines provide answers to the questions asked by user_init file above. In
this case indicating that there are two input-target image pairs to be used in this
training session, the size of the input window, subimage sizes and offsets and the number
of training pairs in an epoch.
$w,2
../images/${sname}.mcid
../images/${sname}_nsl.mcid
$x,$y
0,0
$epoch
../images/${sname}.mcid
../images/${sname}_nsl.mcid
$x,$y
0,104
$epoch
end
```

RUN CONTROL FILE

The following is a typical run control file. It executes the simulation produced by AM, loads the weights from the .save file produced by the learn file above. One forward pass is executed during which the outputs are piped to a short user program nascii2mcid which converts the series of outputs into a .mcid image file. This file is then ftp'ed to the Mac for display.

```
#!/bin/sh
# Run network from saved state and produce image.mcid
set tset=tshapes
set fname=ts1
set  x=506
set  y=209
set w=9
set epoch=`mult $x $y $w`
set sx=`subt $x $w`
set sy=`subt $y $w`
../w/w9h5k3d ${fname}.save << end
$w,1
../images/${tset}.mcid
../images/${tset}.mcid
$x,$y
0,0
$epoch
push sdn
push sdn:Output_Layer
psilent
pnoheader
popenNodes fred nascii2mcid $sx $sy $w ${tset}_${fname}
cycle $epoch
pclose fred
quit
end
ftp -n 130.123.64.44 << end
user RPugmire pwd
binary
put  ${tset}_${fname}.mcid
quit
end
```

DUMP WEIGHTS CONTROL FILE

This control file will dump weights from a previously trained network. The weights are stored in a text file. This file is then transferred to the mac and read into NIH Image to display the weights as a grey level image. The text file can be modified with a text editor to control which weights are to be displayed.

```
#!/bin/sh
set fname=ft7
set x=500
set y=400
set w=15
set epoch=`mult $x $y $w`
../w/w15h0d  -A $fname.save << end >! $fname.wts &
$w,1
../images/targets.mcid
../images/targets.mcid
$x,$y
0,0
$epoch
end
```

APPENDIX C

NIH IMAGE "PLUG-INS" AND USER ROUTINES

This appendix contains the source listings for the Marr-Hildreth edge detection "plug-in'" and the other user routines for NIH Image. These additions and modification to NIH Image were used during the research described in this thesis. The appendix contains the following listings:

- Marr Hildreth Plug-in
- C Utility Routines
- Modifications to NIH Image User.p

MARR-HILDRETH PLUGIN

```
/ *  File: mh.c
     Copyright 1995 by Ralph Pugmire
     Marr Hildreth Edge detection filter to be used with Photoshop or NIH Image and a
     plug-in filter. This routine is separately compiled as a code resource and is
     called by NIH Image at run time. This code requests filter parameters from the
     user using a dialog box, filters the sub-image passed to it and returns the results
     to Image.
* /

// Includes
#include <SetUpA4.h>
#include  "FilterInterface.h"
#include  "Utilities.h"
#include  "mh.h"
#include  "math.h"
#include  <stdlib.h>

// Macro definitions
#define  Length(string)  (*(unsigned  char  *)(string))

// Declarations
typedef struct TParameters{
     int         wsize;
     Boolean     sbsflag,mtflag;
     double      sigma, zband, total;
} TParameters, *PParameters, **HParameters;
//TParameters contains the parameters which are to be stored between calls

short                gResult;
FilterRecordPtr      gStuff;
int                  mx,my,wsize,
                     midx,midy,          // offset to middle of convolving mask
                     xmax,ymax,
                     nxmax,nymax;
Boolean              sbsflag,mtflag;     // flags for output options
double               sigma,              // standard dev. of gaussian mask in pixels
                     zband,
                     total;              // total of mask values. In ideal case =0
imgdr                c_img, m_img;       // my style image descr's for float images
imgd                 i_img,o_img;        // my style image descriptors
/* My style image descriptors provide dynamic two dimensional arrays with a pointer to
the start of each row. Once allocated array elements can be referenced using standard C
notation    fred[5][4]  */
```

```
                    /*****************************/
                    /* Utility routines for the plug-in */
                    /*****************************/
//-------------------------------
//Calculate the total of all pixels in the  mask
double calctotal(Boolean mtflag){
int
    x,y;
double
    zoffset,dtotal,
    dsigma,          /* sigma as double */
     rs,              /* distance from centre of mask*/
    pis4,            /*  1/pi  s^4  */
    ro2s,            /*  r^2 / 2sigma^2  */
    midx,midy;       /* middle of mask */

    midx = (m_img.x - 1)/2.0;
    midy = (m_img.y - 1)/2.0;
    dsigma = sigma;
    pis4 = 1.0 / 3.14159 / pow(dsigma,4.0);
    for (y=dtotal=0; y<m_img.y; y++){
       for (x=0; x<m_img.x; x++){
             rs = (midx-x)*(midx-x) + (midy-y)*(midy-y);
          ro2s = rs / 2 / dsigma / dsigma;
           m_img.e[y][x] = 10 * pis4 * (1.0 - ro2s) * exp(-ro2s);
          dtotal += m_img.e[y][x] ;
       }
    }
    if(mtflag){
       zoffset = (dtotal) / wsize / wsize;
       for (y=dtotal=0; y<m_img.y; y++){
            for (x=0; x<m_img.x; x++){
                    m_img.e[y][x] -= zoffset;
                    dtotal += m_img.e[y][x] ;
          }
       }
    }
    return dtotal;
}

//-------------------------
// Calls the host's TestAbort function
Boolean TestAbort (void){
    return CallPascalB (gStuff->abortProc);
}

//-------------------------------
// Calls the host's UpdateProgress procedure
void UpdateProgress (long done, long total){
```

```
    CallPascal (done, total, gStuff->progressProc);
}

//-------------------------------
// Centers a dialog template 1/3 of the way down on the main screen.
#define menuHeight 20
void CenterDialog (DialogTHndl dt){
    Rect r;
    short width;
    short height;
    width  = screenBits.bounds.right;
    height = screenBits.bounds.bottom;
    r = (**dt).boundsRect;
    OffsetRect (&r, -r.left, -r.top);
    OffsetRect (&r, (width - r.right) / 2,
                    (height - r.bottom - menuHeight) / 3 + menuHeight);
    (**dt).boundsRect = r;
}
#undef menuHeight

//-------------------------------------
// Displays the about dialog box for the plug-in module.
void DoAbout (void){
    short item;
    DialogPtr dp;
    DialogTHndl dt;
    dt = (DialogTHndl) GetResource ('DLOG', AboutdialogID);
    HNoPurge ((Handle) dt);
    CenterDialog (dt);
    dp = GetNewDialog (AboutdialogID, nil, (WindowPtr) -1);
    ModalDialog (nil, &item);
    DisposDialog (dp);
    HPurge ((Handle) dt);
}
#undef dialogID

//-----------------------------------
/* UserItem to outline the OK button in a dialog box.
pascal void OutlineOK (DialogPtr dp, short item){
    Rect r;
    Handle h;
    short itemType;
    SetUpA4 ();
    item = OK;
    GetDItem (dp, item, &itemType, &h, &r);
    PenNormal ();
    PenSize (3, 3);
    InsetRect (&r, -4, -4);
    FrameRoundRect (&r, 16, 16);
```

```
    PenNormal ();
    RestoreA4 ();
}

//-----------------------------------------------------------
/* Asks the user for the plug-in filter module's parameters.
   The image size information is not yet defined at this point.
   The calling program may not call this routine every time the
   filter is run (it may save the data held by the parameters handle in
   a macro file). */
void DoParameters (void){
    Rect r;
    short j;
    Str255 s;
    unsigned char *p, *pp;
    Handle h;
    short item;
    DialogPtr dp;
    short itemType;
    DialogTHndl dt;
    int x,y;
    float prevtotal;
    Boolean afterpeak;

    if (!gStuff->parameters){
       gStuff->parameters = NewHandle ((long) sizeof (TParameters));
       if (!gStuff->parameters)
              {
              gResult = memFullErr;
              return;
              }
       ((PParameters) *gStuff->parameters)->sigma = 1.2;
       ((PParameters) *gStuff->parameters)->zband = 0.01;
       ((PParameters) *gStuff->parameters)->total = 0.0;
       ((PParameters) *gStuff->parameters)->wsize = 11;
       ((PParameters) *gStuff->parameters)->mtflag = false;
       ((PParameters) *gStuff->parameters)->sbsflag = true;
    }
    dt = (DialogTHndl) GetResource ('DLOG', ParamsdialogID);
    HNoPurge ((Handle) dt);
    CenterDialog (dt);
    dp = GetNewDialog (ParamsdialogID, nil, (WindowPtr) -1);
    RememberA4 ();

    // Initialize filter parameters to be used in the dialog box
    GetDItem (dp, hookItem, &itemType, &h, &r);
    SetDItem (dp, hookItem,  itemType, (Handle) &OutlineOK, &r);
    sigma = ((PParameters) *gStuff->parameters)->sigma;
    zband = ((PParameters) *gStuff->parameters)->zband;
```

```
wsize = ((PParameters) *gStuff->parameters)->wsize;
mtflag = ((PParameters) *gStuff->parameters)->mtflag;
sbsflag = ((PParameters) *gStuff->parameters)->sbsflag;
gResult = dimimgr(&m_img,wsize,wsize); /* create and init mask image */
if (gResult != noErr) return;
total = calctotal(mtflag);
freeimgr(&m_img);
SetDReal(dp,sigmaItem,sigma,2);
SetDReal(dp,zbandItem,zband,3);
SetDReal(dp,totalItem,total,6);
SetDNum(dp,wsizeItem,wsize);
SetDialogItem(dp,mtflagItem,mtflag);
SetDialogItem(dp,sbsflagItem,sbsflag);
SelIText (dp, sigmaItem, 0, 32767);

//Display dialog box and process user input
do{
    ModalDialog (nil, &item);
    switch (item){
    case wsizeItem: // Check window size entered
            /* Beep and remove anything other than 0-9 */
            GetDString(dp,wsizeItem,s);
            for (p=s+1; p<=s+s[0]; p++){
                    if (*p>'9' || *p<'0'){
                            SysBeep(1);
                            s[0]--;
                            for (pp=p; pp<=s+s[0]; pp++)
                                    *pp = *(pp+1);
                    }
                    SetDString(dp,wsizeItem,s);
            }
            break;
    case calcwsizeItem://Calculate window size from sigma
            //Find odd value for which total is less than 0.01 after reaching a max
            // Note we need to create temporary mask to calc total
            sigma = GetDReal(dp,sigmaItem);
            afterpeak = false;
            for (wsize=3; wsize<25; wsize += 2){
                    SetDNum(dp,wsizeItem,wsize);
                    /* create and init a mask */
                    gResult = dimimgr(&m_img,wsize,wsize);
                    if (gResult != noErr) return;
                    total = calctotal(false);
                    SetDReal(dp,totalItem,total,6);
                    freeimgr(&m_img);
                    if (total<prevtotal)  afterpeak=true;
                    if ((total<0.01) && afterpeak) break;
                    prevtotal = total;
            }
```

```
            gResult = dimimgr(&m_img,wsize,wsize); // create and init mask img
            if (gResult != noErr) return;
            total = calctotal(mtflag);
            SetDReal(dp,totalItem,total,6);
            freeimgr(&m_img);
            break;

        case sbsflagItem:// Set or reset scale by size
            sbsflag = !sbsflag;
            SetDialogItem(dp,sbsflagItem,sbsflag);
            break;

        case mtflagItem:// Set or reset scale by distance
            mtflag = !mtflag;
            SetDialogItem(dp,mtflagItem,mtflag);
            if (!mtflag)
                SelIText (dp, zbandItem, 0, 32767);
            else
                SelIText (dp, sigmaItem, 0, 32767);
            // Then fall into calcItem

        case calcItem://Calculate total of pixel values in mask
            sigma = GetDReal(dp,sigmaItem);
            wsize = GetDNum(dp,wsizeItem);
            gResult = dimimgr(&m_img,wsize,wsize); //create and init mask image
            if (gResult != noErr) return;
            total = calctotal(mtflag);
            SetDReal(dp,totalItem,total,6);
            freeimgr(&m_img);
            break;
        }
    }while (item != OK && item != Cancel);
    if (item == OK){
        sigma = GetDReal(dp,sigmaItem);
        wsize = GetDNum(dp,wsizeitem);
        zband = GetDReal(dp,zbandItem);
        ((PParameters) *gStuff->parameters)->sigma = sigma;
        ((PParameters) *gStuff->parameters)->zband = zband;
        ((PParameters) *gStuff->parameters)->total = total;
        ((PParameters) *gStuff->parameters)->wsize = wsize;
        ((PParameters) *gStuff->parameters)->mtflag = mtflag;
        ((PParameters) *gStuff->parameters)->sbsflag = sbsflag;
    }
    DisposDialog (dp);
    HPurge ((Handle) dt);
    if (item == Cancel){
        gResult = 1;
        return;
    }
```

```
}
#undef dialogID
#undef hookItem

//----------------------
/* Prepare to filter an image. */
void DoPrepare (void){
    typedef short double *psdouble;
    double zoffset;
    int  i,x,y;
    unsigned char *ptr;

    sigma = ((PParameters) *gStuff->parameters)->sigma;
    zband = ((PParameters) *gStuff->parameters)->zband;
    wsize = ((PParameters) *gStuff->parameters)->wsize;
    mtflag = ((PParameters) *gStuff->parameters)->mtflag;
    sbsflag = ((PParameters) *gStuff->parameters)->sbsflag;
    my = gStuff->filterRect.bottom - gStuff->filterRect.top + 1;
    mx = gStuff->filterRect.right - gStuff->filterRect.left + 1;
    xmax = mx - wsize + 1;
    ymax = my - wsize + 1;
    midx = (wsize - 1)/2.0;
    midy = (wsize - 1)/2.0;
    nxmax = mx - midx -1;
    nymax = my - midy -1;
    /* tell image how much space we want */
    gStuff->bufferSpace =    (long)wsize * wsize * sizeof(short double)
                                + (long)wsize * sizeof(psdouble)
                                +(long)mx * my * sizeof(short double)
                                + (long)my * sizeof(psdouble);
}

//-----------------------------------------------
/* Requests pointer to the first part of the image to be filtered. */
/* Ask for all the image in one go */
void DoStart (void){
gResult = dimimgr(&c_img,my,mx); /* create and init real conv img*/
    if (gResult != noErr) return;
    gResult = dimimgr(&m_img,wsize,wsize); /* create and init kernel image */
    if (gResult != noErr){
       freeimgr(&c_img);
       return;
    }
    total = calctotal(mtflag);
    gStuff->inRect = gStuff->filterRect;
    gStuff->inLoPlane = 0;
    switch (gStuff->imageMode){
       case filterModeGrayScale:
              gStuff->inHiPlane = 0;
```

```
            break;
        default:
            gResult = filterBadMode;// we only handle greyscale images
    }
    gStuff->outRect = gStuff->inRect;
    gStuff->outLoPlane = gStuff->inLoPlane;
    gStuff->outHiPlane = gStuff->inHiPlane;
}

//-----------------------------------
// Filters the area
void DoContinue (void){
    #define max(a,b) ((a>b) ? (a) : (b))
    #define min(a,b) ((a<b) ? (a) : (b))
    int
        i,d,x,y,sj,j,jj;
    short double
        *re,smin,smax,cmin,cmax,
        c,mzband;
    unsigned char
        *tpe,*pe,*ptr;

    /*convert the input and output roi's to my style images*/
    i_img.y = my;
    i_img.x = mx;
    if (!(i_img.e = (ppc)NewPtr(sizeof(ptr) * my))){
        gResult = memFullErr; return;}
    ptr=(unsigned char *) gStuff->inData;
    for (i=0; i<my; i++){
        i_img.e[i] = ptr;
        ptr += gStuff->inRowBytes;
    }
    o_img.y = my;
    o_img.x = mx;
    if (!(o_img.e = (unsigned char **)NewPtr(sizeof(ptr) * my))){
        gResult = memFullErr; return;}
    ptr=(unsigned char *) gStuff->outData;
    for (i=0; i<my; i++){
        o_img.e[i] = ptr;
        ptr += gStuff->outRowBytes;
    }
    mzband = zband * 255.0;

    //Convolve mask with input image
      for (y=0;y < ymax; y++){
        UpdateProgress ((long) y,(long) ymax+nymax/10.0);
        if (TestAbort ()){
            gResult = 1;
            return;
```

```
    }
    for (x=0; x < xmax; x++){
        sj = y + wsize;
          for (jj=0,j=y,c=0; j < sj ; j++,jj++){
                re = m_img.e[jj];
                pe = &i_img.e[j][x];
                tpe = &i_img.e[j][x + wsize];
                for (; pe < tpe; c += *re++ * *pe++);
        }
        if (c>0 && c<=mzband)
                c=0;
        if (c<0 && c>=mzband)
                c=0;
         c_img.e[y+midy][x+midx] = c;
    }
}

/*Detect zero crossings */
 for (y=midy+1,smax=0; y < nymax-1; y++){
    UpdateProgress ((long) ymax+y/10,(long) ymax+nymax/10.0);
    for (x=midx+1; x < nxmax-1; x++){
        short double a,b,c,s;
         c = c_img.e[y][x];
         if (c<0) c = -c;
        s = 0;
          a = c_img.e[y-1][x-1];
          b = c_img.e[y+1][x+1];
         if (a>0 && b<0) s = max(s,a-b);
         else if (a<0 && b>0) s = max(s,b-a);
          a = c_img.e[y-1][x];
          b = c_img.e[y+1][x];
         if (a>0 && b<0) s = max(s,a-b);
         else if (a<0 && b>0) s = max(s,b-a);
          a = c_img.e[y-1][x+1];
          b = c_img.e[y+1][x-1];
         if (a>0 && b<0) s = max(s,a-b);
         else if (a<0 && b>0) s = max(s,b-a);
          a = c_img.e[y][x-1];
          b = c_img.e[y][x+1];
         if (a>0 && b<0) s = max(s,a-b);
         else if (a<0 && b>0) s = max(s,b-a);
          if (s!=0){
              if (c>s) s=0;
              else if (sbsflag) s = s - c; else s = 1 - c/s;
        }
         smax = max(s,smax);
          c_img.e[y-2][x-2] = s;
    }
}
```

```
    /* put results in the ouput image and scale to 0-255 */
     for (y=midy+1; y < nymax-1; y++)
         for (x=midx+1; x < nxmax-1; x++)
              o_img.e[y][x] = 255.0 - (c_img.e[y-2][x-2]*255.0/smax);
     SetRect (&gStuff->inRect, 0, 0, 0, 0); /* let em know we finished */
    gStuff->outRect = gStuff->inRect;
}

//-------------------------------------------------
/* This routine will always be called if DoStart does not return an error
   (even if DoContinue returns an error or the user aborts the operation).
   This allows the module to perform any needed cleanup. */
void DoFinish (void){
    DisposPtr((Ptr)i_img.e);    /*Free dynamic space used by img desc array of pts*/
    DisposPtr((Ptr)o_img.e);    /*Free dynamic space used by img desc array of pts*/
      freeimgr(&m_img);
      freeimgr(&c_img);
}

                              /*****************************/
                              /* Main dispatching routine. */
                              /*****************************/
/* Initializes and sets up the global variables,and performs the operation specified by
the selector. This is the routine actually called by Photoshop or NIH Image */
pascal void main (short selector, FilterRecordPtr stuff, long *data, short *result){
    /* Allow access to globa! variables */
    RememberA0 ();
    SetUpA4 ();
    /* Copy the current quickdraw globals into the plug-in local copy */
    asm
      {
            MOVE.L      (A5),A0 ; Get address of real quickdraw globals
            SUB.L  #126,A0      ; Move to start
            LEA    randSeed,A1  ; Get address of local copy
            MOVE.W      #64,D0              ; Globals are 65 words long
      @1    MOVE.W      (A0)+,(A1)+         ; Copy a word
            DBF    D0,@1                ; Move to next word
      }
    /* Perform the requested operation */
    gStuff     = stuff;
    gResult = noErr;
    switch  (selector)
      {
      case filterSelectorAbout:
            DoAbout ();
            break;
      case filterSelectorParameters:
            DoParameters ();
```

```
            break;
        case filterSelectorPrepare:
            DoPrepare ();
            break;
        case filterSelectorStart:
            DoStart ();
            break;
        case filterSelectorContinue:
            DoContinue ();
            break;
        case filterSelectorFinish:
            DoFinish ();
            break;
        default:
            gResult = filterBadParameters;
        }
    *result = gResult;
    /* Restore the application's A4 register */
    RestoreA4 ();
}
```

UTILITY ROUTINES

This is the listing of the general C utilities used in the plugin

```
// Utilities.c
// R.H. Pugmire
// My general utilitiy routines
#include <Types.h>
#include <stdlib.h>
#include <stdio.h>
#include "Utilities.h"
#include "math.h"

typedef short double *psdouble, **ppsdouble;

// Routine to get a line of chars <100
// Return nchar = # we got
OSErr getline(short refNum,char *s,int max){
        OSErr err;
        int     i;
        long    count;
        for(i=0; i<max; i++){
                        count = 1;
                        err = FSRead(refNum,&count,&s[i]);
                        if (err) {SysBeep(2);FSClose(refNum);return err;}
                        if (s[i] == 13) {
                                if (s[0] == '!'){
                                        i=-1;
                                }else{
                                        s[i] = 0;
                                        return noErr;
                                }
                        }
        }
        return;
}

// Routine to dimension one of my image descriptors for a real image
OSErr dimimgr(Pimgdr m, int y, int x){
        int i;
        m->y = y;
        m->x = x;
        if (!(m->e = (ppsdouble) NewPtr(sizeof(psdouble) * (long)y)))
                return memFullErr;
        for (i=0; i<y; i++){
                if (!(m->e[i] = (psdouble)NewPtr(sizeof(short double) * (long)x))){
```

```
                    m->y = i;
                    freeimgr(m);
                    return  memFullErr;
                }
        }
        return  noErr;
}

//Routine used to free the memory used in one of my style images
void  freeimgr(Pimgdr  m){
        int  y,i;
        y = m->y;
        for  (i=0;  i<y;  i++)
                DisposPtr((Ptr)m->e[i]);
        DisposPtr((Ptr)m->e);
}

// Get value of an item in a dialog box
int  GetDialogItem(DialogPtr  TheDialog,  int  item){
        short  ItemType;
        Rect ItemBox;
        Handle ItemHdl;
        Str255  str;
        GetDItem(TheDialog,  item,  &ItemType,  &ItemHdl,  &ItemBox);
        return  GetCtlValue((ControlHandle)ItemHdl);
}

// Set the value of an item in a dialog box
void  SetDialogItem(DialogPtr  TheDialog,  int  item,  int  n){
    short ItemType;
    Rect ItemBox;
    Handle ItemHdl;
     Str255 str;
     GetDItem(TheDialog,  item,  &ItemType,  &ItemHdl,  &ItemBox);
     SetCtlValue((ControlHandle)ItemHdl,  n);
}

// Put an integer into a field in a dialog box
void  SetDNum(DialogPtr  TheDialog,  int  item,  int  n){
    short ItemType;
    Rect ItemBox;
    Handle ItemHdl;
     Str255 str;
     GetDItem(TheDialog,  item,  &ItemType,  &ItemHdl,  &ItemBox);
      NumToString(n,str);
      SetIText(ItemHdl,  str);
}

// Put a float into a field in a dialog box
```

```
void SetDReal(DialogPtr TheDialog, int item, double n, int fwidth){
        short ItemType;
        Rect ItemBox;
        Handle ItemHdl;
        Str255 str;
        GetDItem(TheDialog, item, &ItemType, &ItemHdl, &ItemBox);
        Real2String(n,1, fwidth, str);
        SetIText(ItemHdl, str);
}

// Get a float from a field in a dialog box
double GetDReal(DialogPtr TheDialog, int item){
        Str255 str;
        Rect r;
        short itemType;
        Handle Text;
        GetDItem (TheDialog, item, &itemType, &Text, &r);
        GetIText (Text, str);
        return String2Real(str);
}

//Get an int from a field in a dialog box
int GetDNum(DialogPtr TheDialog, int item){
        long n;
        Str255 str;
        Rect r;
        short itemType;
        Handle Text;
        GetDItem (TheDialog, item, &itemType, &Text, &r);
        GetIText (Text, str);
        StringToNum(str,&n);
        return n;
}

// Put a string into a field in a dialog box
void SetDString(DialogPtr TheDialog, int item, Str255 str){
        short ItemType;
        Rect ItemBox;
        Handle ItemHdl;
        GetDItem(TheDialog, item, &ItemType, &ItemHdl, &ItemBox);
        SetIText(ItemHdl, str);
}

// Get a string from a field in a dialog box
void GetDString(DialogPtr TheDialog, int item, Str255 str){
        Rect r;
        short itemType;
        Handle Text;
        GetDItem (TheDialog, item, &itemType, &Text, &r);
```

```
        GetIText (Text, str);
}

// Return the value of a printed floating point number contained in a string
double String2Real(Str255 str){
        PtoCstr(str);
        return atof((char *)str);
}

 // Convert a floating point number into a printable string
void Real2String(double val, int width, int fwidth, Str255 str){
        char fs[] = "%6.2f";
        /* fwidth =-1 => make up a sensible value for fwidth  0 2 or 4 */
        if ( fwidth < 0 ) {
            if ( val < 1.0 )
                fwidth = 4;
             else if ( (int)val == val )
                fwidth = 0;
           else
                fwidth = 2;
        }
        fs[1] = '0'+width;
        fs[3] = '0'+fwidth;
        sprintf((char *)str,fs,val);
        CtoPstr((char *)str);
 }
```

USER.P FROM NIH IMAGE

In addition to macros and plug-ins NIH image can be customized by adding code to the user.p file and recompiling the source. The following is the listing of the modified version of user.p which contains routines for image comparison, addition of signal dependant noise and addition of gaussian noise.

```
unit User;
{Modified sections from user.p in NIH Image}

interface
 uses
   Script, QuickDraw, Palettes, PrintTraps, globals, Utilities, Functions, Graphics,
AppleTalk;
 procedure InitUser;
 procedure DoUserMenuEvent (MenuItem: integer);
var {I think these are globals which can be seen from other units ie global globals }
  fred: integer;
implementation
 const
  PixelsPerUpdate = 600;
 type
   rvector = array[1..1000] of real;
 var
  color: integer;
{RHP Globals }
 mymenu: MenuHandle;
  err: OSErr;

function mmmax (a, b: real): real;
 begin
  if a > b then
   mmmax := a
 else
  mmmax := b
 end;

 function signof (a: real): integer;
 begin
  if a > 0 then
   signof := 1
 else
   signof := -1
 end;
```

```
 procedure InitUser;
 begin
  UserMenuH := GetMenu(UserMenu);
   InsertMenu(UserMenuH, 0);

  DrawMenuBar;
  {Additional user initialization code goes here.}
  RHPInit;
 end;

 procedure DoUserCommand1;
  var
  ok: boolean;
 begin
end;

 procedure DoUserCommand2;
  var
  ok: boolean;
 begin
end;

{   -------------------------RHP   User   routines   start   here   ------------}

{ FUnction to add two images together}
function ImageAdd: boolean;
  type
    fred = packed array[1..100000] of integer;
    fredptr = ^fred;
  var
    src1, src2, dst: ptr;
   Line1, Line2: LineType;
    inp1info, inp2info, OutInfo, TempInfo: InfoPtr;
   s1width, s2width, mx, my, n, i, c: Longint;
   image1, image2, name: str255;
    itemhit: integer; {the itemnumber they select}
   mydialogid: integer;
   pmydialog: DialogPtr;
    fredp: fredptr;
   LinesPerUpdate, LineCount, min, max, temp: integer;
    StartTicks: LongInt;
   modifiedRect: Rect;
    m1, m2, mark, NewMark: Integer;
  scale: boolean;
 begin
 ImageAdd := false;
  StartTicks := TickCount;
  mydialogid := 129;
   while CountMItems(mymenu) > nPics do begin
```

```
    DelMenuItem(mymenu, 1);
  end;
  while CountMItems(mymenu) < nPics do begin
    AppendMenu(mymenu, 'dummy');
  end;
  for i := 1 to nPics do begin
     TempInfo := pointer(WindowPeek(PicWindow[i])^.RefCon);
     SetItem(mymenu, i, TempInfo^.title);
  end;
   InsertMenu(mymenu, -1);
  pmydialog := GetNewDialog(mydialogid, nil, POINTER(-1));
  Checkon(pmydialog, 4);
  SetDialogItem(pmydialog, 5, Info^.PicNum);
  SetDialogItem(pmydialog, 6, Info^.PicNum - 1);
  repeat
    ModalDialog(nil, itemhit);
   if itemhit = 4 then
     SetDialogItem(pmydialog, 4, ord(not Check(pmydialog, 4)));
   until (itemhit = ok) or (itemhit = cancel);
  if itemhit = cancel then begin
    DisposDialog(pmydialog);
    DeleteMenu(134);
   exit(ImageAdd);
  end;
  scale := Check(pmydialog, 4);
  m1 := GetCtl(pmydialog, 5);
  m2 := GetCtl(pmydialog, 6);
  DisposDialog(pmydialog);
   DeleteMenu(134);
  for i := 1 to nPics do begin
    if i = m1 then begin
       Inp1Info := pointer(WindowPeek(PicWindow[i])^.RefCon);
      image1 := Inp1Info^.title;
    end;
    if i = m2 then begin
       Inp2Info := pointer(WindowPeek(PicWindow[i])^.RefCon);
      image2 := Inp2Info^.title;
    end;
  end;
  name := image1;
   if (length(image1) + length(image2)) > 31 then begin
     delete(name, 17, length(name) - 16);
    name := concat(name, '+', image2);
     delete(name, 33, length(name) - 32);
  end
  else
    name := concat(image1, '+', image2);
s1width := Inp1Info^.BytesPerRow;
  s2width := Inp2Info^.BytesPerRow;
```

```
if (s2width > s1width) or (Inp2Info^.nlines > Inp1Info^.nlines) then begin
 beep;
  PutMessage('Second image cannot be bigger than the first one');
  exit(ImageAdd);
end;
 mx := Inp2Info^.PixelsPerLine;
 my := Inp2Info^.nlines;
LinesPerUpdate := PixelsPerUpdate div mx;
if NewPicWindow(name, mx, my) = false then begin
 beep;
  PutMessage('Not enough memory to create an image');
  exit(ImageAdd);
end;
 fredp := fredptr(NewPtr(SIZEOF(integer) * mx * my));
if fredp = nil then begin
 beep;
  PutMessage('Not enough memory to create int array');
  exit(ImageAdd);
end;
ShowWatch;
max := 0;
 min := 512;
with Info^ do begin
  src1 := Inp1Info^.PicBaseAddr;
  src2 := Inp2Info^.PicBaseAddr;
 dst := PicBaseAddr;
  mark := PicRect.Top;
  for i := 0 to my - 1 do begin
     BlockMove(src1, @Line1, mx);
       src1 := ptr(ord4(src1) + s1width);
     BlockMove(src2, @Line2, mx);
       src2 := ptr(ord4(src2) + s2width);
   c := 0;
   while c < mx do begin
      temp := Line1[c] + Line2[c];
    if temp > max then
    max := temp;
    if temp < min then
    min := temp;
     fredp^[c + i * mx] := temp;
    c := c + 1;
   end;
  end;
  max := max - min;
    ShowMessage(concat('Min,Max= ', StringOf(min), StringOf(max)));
  for i := 0 to my - 1 do begin
   c := 0;
   while c < mx do begin
    if scale then
```

```
        Line1[c] := Round((fredp^[c + i * mx] - min) / max * 255.0)
      else begin
        temp := fredp^[c + i * mx];
       if temp > 255 then
        Line1[c] := 255
      else
        Line1[c] := temp;
      end;
       c := c + 1;
      end;
       BlockMove(@Line1, dst, mx);
        dst := ptr(ord4(dst) + mx);
      LineCount := LineCount + 1;
      if LineCount >= LinesPerUpdate then begin
       LineCount := 0;
       {Try and update only the region we have drawn in}
       NewMark := PicRect.Top + i;
        SetRect(modifiedRect, PicRect.left, mark, PicRect.Right, NewMark);
       UpdateScreen(modifiedRect);
       mark := NewMark;
         ShowMessage(concat('Line ', StringOf(i)));
      if CommandPeriod then begin
      beep;
       exit(ImageAdd)
      end;
      end;
    end;
   end; {with}
  ShowTime(StartTicks, Info^.PicRect, '');
 if fredp <> nil then
     DisposPtr(Ptr(fredp));
 ImageAdd := true;
end;

{Add signal dependent noise to an image}
function Sdn (name: str255; SavingBlankField: boolean): boolean;
  var
    width, height, hstart, vstart, i, c: integer;
   SaveInfo: InfoPtr;
   src, dst: ptr;
   offset: LongInt;
   AutoSelectAll: boolean;
   Line: LineType;
   n, r1, s, t, sx: Real;
   LinesPerUpdate, LineCount, mark, NewMark: Integer;
   StartTicks: LongInt;
  ModifiedRect: Rect;
begin
 Sdn := false;
```

```
StartTicks := TickCount;
if nPics = MaxPics then
  exit(Sdn);
WhatToUndo := NothingToUndo;
if (not SavingBlankField) and (NotRectangular or NotinBounds) then
  exit(Sdn);
AutoSelectAll := (not Info^.RoiShowing) or SavingBlankField;
if AutoSelectAll then
  SelectAll(false);
ShowWatch;
with info^ do begin
  if name = '' then begin
    name := concat(title, ' with sdnsp');
    if length(name) > 32 then
      delete(name, 33, length(name) - 32);
  end;
  with RoiRect do begin
    width := right - left;
    if odd(width) then begin
      if (left + width < PicRect.right) then
      width := Width + 1
    else
      Width := width - 1;
    end;
    height := bottom - top;
    hstart := left;
    vstart := top;
  end;
end;
if AutoSelectAll then
  KillRoi;
SaveInfo := Info;
if NewPicWindow(name, width, height) then
with SaveInfo^ do begin
  LinesPerUpdate := PixelsPerUpdate div width;
  mark := Info^.RoiRect.top;
  offset := LongInt(vstart) * BytesPerRow + hstart;
  src := ptr(ord4(PicBaseAddr) + offset);
  dst := Info^.PicBaseAddr;
  for i := 0 to height - 1 do begin
    BlockMove(src, @Line, width);
    src := ptr(ord4(src) + BytesPerRow);
    c := 0;
    while c < width do begin
      s := Line[c];
      r1 := Random / 65534.0 + 0.5;
      while r1 = 0 do
        r1 := Random / 65534.0 + 0.5;
      t := sqrt(ln(1 / r1 / r1));
```

```
        t := (2.515517 + 0.802853 * t + 0.010328 * t * t) / (1 + 1.432788 * t +
        0.189269 * t * t + 0.001308 * t * t * t) - t;
       sx := s + (sqrt(s) + 15) * t;
      if sx > 255 then
      sx := 255
      else if sx < 0 then
      sx := 0;
       if Random >= (0.97 * 32767) then
      sx := 0
       else if Random >= (0.97 * 32767) then
      sx := 255;
       Line[c] := Round(sx);
      c := c + 1;
     end;
       BlockMove(@Line, dst, width);
        dst := ptr(ord4(dst) + width);
      LineCount := LineCount + 1;
      if LineCount >= LinesPerUpdate then begin
      LineCount := 0;
      {Try and update only the region we have drawn in}
      NewMark := PicRect.Top + i;
       SetRect(modifiedRect, PicRect.left, mark, PicRect.Right, NewMark);
      UpdateScreen(modifiedRect);
      mark := NewMark;
        ShowMessage(concat('Line ', StringOf(i)));
     if CommandPeriod then begin
     beep;
      exit(Sdn)
     end;
     end;
    end;
    if SavingBlankField then begin
      Info^.PIctureType := BlankField;
      BlankFieldInfo := info;
    end;
     ShowTime(StartTicks, Info^.PicRect, '');
    Sdn := true;
   end; {with}
end;

{Add Gaussian noise to an image}
function Noise (name: str255; SavingBlankField: boolean): boolean;
  var
    width, height, hstart, vstart, i, c: integer;
   SaveInfo: InfoPtr;
   src, dst: ptr;
  offset: LongInt;
  AutoSelectAll: boolean;
  Line: LineType;
```

```
  n, r1, s, t, sx: Real;
  LinesPerUpdate, LineCount, mark, NewMark: Integer;
  StartTicks: LongInt;
 ModifiedRect: Rect;
begin
 Noise := false;
 StartTicks := TickCount;
 if nPics = MaxPics then
   exit(Noise);
 WhatToUndo := NothingToUndo;
 if (not SavingBlankField) and (NotRectangular or NotinBounds) then
   exit(Noise);
 AutoSelectAll := (not Info^.RoiShowing) or SavingBlankField;
 if AutoSelectAll then
   SelectAll(false);
 ShowWatch;
 with info^ do begin
  if name = '' then begin
     name := concat(title, ' with sdnsp');
    if length(name) > 32 then
      delete(name, 33, length(name) - 32);
  end;
  with RoiRect do begin
     width := right - left;
    if odd(width) then begin
      if (left + width < PicRect.right) then
     width := Width + 1
    else
      Width := width - 1;
    end;
    height := bottom - top;
    hstart := left;
    vstart := top;
  end;
 end;
 if AutoSelectAll then
   KillRoi;
 SaveInfo := Info;
 if NewPicWindow(name, width, height) then
 with SaveInfo^ do begin
   LinesPerUpdate := PixelsPerUpdate div width;
   mark := Info^.RoiRect.top;
    offset := LongInt(vstart) * BytesPerRow + hstart;
    src := ptr(ord4(PicBaseAddr) + offset);
  dst := Info^.PicBaseAddr;
   for i := 0 to height - 1 do begin
     BlockMove(src, @Line, width);
      src := ptr(ord4(src) + BytesPerRow);
    c := 0;
```

```
      while c < width do begin
       s := Line[c];
        r1 := Random / 65534.0 + 0.5;
       while r1 = 0 do
        r1 := Random / 65534.0 + 0.5;
          t := sqrt(ln(1 / r1 / r1));
          t := (2.515517 + 0.802853 * t + 0.010328 * t * t) / (1 + 1.432788 * t +
                0.189269 * t * t + 0.001308 * t * t * t) - t;
       sx := s + 32 * t;
       if sx > 255 then
       sx := 255
       else if sx < 0 then
       sx := 0;
        Line[c] := Round(sx);
       c := c + 1;
      end;
        BlockMove(@Line, dst, width);
         dst := ptr(ord4(dst) + width);
       LineCount := LineCount + 1;
       if LineCount >= LinesPerUpdate then begin
       LineCount := 0;
       {Try and update only the region we have drawn in}
       NewMark := PicRect.Top + i;
        SetRect(modifiedRect, PicRect.left, mark, PicRect.Right, NewMark);
       UpdateScreen(modifiedRect);
       mark := NewMark;
         ShowMessage(concat('Line ', StringOf(i)));
      if CommandPeriod then begin
     beep;
       exit(Noise)
     end;
     end;
    end;
    if SavingBlankField then begin
      Info^.PIctureType := BlankField;
      BlankFieldInfo := info;
    end;
      ShowTime(StartTicks, Info^.PicRect, '');
    Noise := true;
   end; {with}
end;

{Function for finding three peaks used in rule for finding e's}
function threepeaks (name: str255; SavingBlankField: boolean): boolean;
 var
   width, height, hstart, vstart, i, c: integer;
  SaveInfo: InfoPtr;
  psrc, src, dst: ptr;
 offset: LongInt;
```

```
  cncld, goingup, AutoSelectAll: boolean;
  Line: LineType;
  vs, rp, s, p, pp, ti, npeaks: Integer;
  LinesPerUpdate, LineCount, mark, NewMark: Integer;
  StartTicks: LongInt;
 ModifiedRect: Rect;
begin
 threepeaks := false;
 StartTicks := TickCount;
 if nPics = MaxPics then
   exit(threepeaks);
 WhatToUndo := NothingToUndo;
 if (not SavingBlankField) and (NotRectangular or NotinBounds) then
   exit(threepeaks);
 AutoSelectAll := (not Info^.RoiShowing) or SavingBlankField;
 vs := GetInt('Enter scan height', 7, cncld) div 2;
 if cncld then
   exit(threepeaks);
 rp := GetInt('Enter number of peaks to look for', 3, cncld);
 if cncld then
   exit(threepeaks);
 if AutoSelectAll then
   SelectAll(false);
 ShowWatch;
 with info^ do begin
  if name = '' then begin
     name := concat(title, ' peaks');
     if length(name) > 32 then
       delete(name, 33, length(name) - 32);
   end;
  with RoiRect do begin
     width := right - left;
     if odd(width) then begin
       if (left + width < PicRect.right) then
      width := Width + 1
     else
       Width := width - 1;
     end;
     height := bottom - top;
     hstart := left;
     vstart := top;
   end;
 end;
 if AutoSelectAll then
   KillRoi;
 SaveInfo := Info;
 if NewPicWindow(name, width, height) then
 with SaveInfo^ do begin
    LinesPerUpdate := PixelsPerUpdate div width;
```

```
 mark := Info^.RoiRect.top;
  offset := LongInt(vstart) * BytesPerRow + hstart;
  src := ptr(ord4(PicBaseAddr) + offset);
dst := Info^.PicBaseAddr;
 for i := 0 to height - 1 do begin
    BlockMove(src, @Line, width);
     src := ptr(ord4(src) + BytesPerRow);
  c := 0;
  while c < width do begin
   s := Line[c];
  npeaks := 0;
  goingup := true;
  pp := 0;
   for ti := -vs to +vs do begin
     psrc := ptr(ord4(src) - BytesPerRow * ti + c);
   p := BAND(psrc^, 255);
  if (p < pp) and goingup then begin
  goingup := false;
  npeaks := npeaks + 1;
 end
  else if p > pp then
  goingup := true;
  pp := p;
 end;
  if npeaks = rp then
   Line[c] := 255
 else
  Line[c] := s;
 c := c + 1;
 end;
   BlockMove(@Line, dst, width);
    dst := ptr(ord4(dst) + width);
 LineCount := LineCount + 1;
 if LineCount >= LinesPerUpdate then begin
 LineCount := 0;
 {Try and update only the region we have drawn in}
 NewMark := PicRect.Top + i;
   SetRect(modifiedRect, PicRect.left, mark, PicRect.Right, NewMark);
 UpdateScreen(modifiedRect);
 mark := NewMark;
   ShowMessage(concat('Line ', StringOf(i)));
 if CommandPeriod then begin
 beep;
  exit(threepeaks)
 end;
 end;
end;
if SavingBlankField then begin
  Info^.PIctureType := BlankField;
```

```
      BlankFieldInfo := info;
    end;
     ShowTime(StartTicks, Info^.PicRect, '');
    threepeaks := true;
   end; {with}
end;

{Compare two images and display various similarity measures}
function ImageDiff: boolean;
const
  dborder = 11;
  dxoffset = 12;
  dyoffset = 13;
  dxsize = 14;
  dysize = 15;
var
   src1, src2: ptr;
  Line1, Line2: LineType;
   inp1info, inp2info, TempInfo: InfoPtr;
  w1, w2, mx1, my1, mx2, my2, n, i, c: Longint;
  str, str1, image1, image2, name: str255;
  itemhit: integer; {the itemnumber they select}
 mydialogid: integer;
 pmydialog: DialogPtr;
 min, max, temp: integer;
 ae, me, mae, mse, msne, rms, Per, Pm, pc, pnc: real;
 m1, m2, diff, a, b, np, nzp, lt, gt: Longint;
  d, xsize, ysize, fxsize, fysize, xoffset, yoffset, border, StartTicks: LongInt;
begin
 ImageDiff := false;
 StartTicks := TickCount;
 mydialogid := 132;
 str := '';
 while CountMItems(mymenu) > nPics do begin
   DelMenuItem(mymenu, 1);
 end;
 while CountMItems(mymenu) < nPics do begin
   AppendMenu(mymenu, 'dummy');
 end;
 for i := 1 to nPics do begin
    TempInfo := pointer(WindowPeek(PicWindow[i])^.RefCon);
    SetItem(mymenu, i, TempInfo^.title);
 end;
  InsertMenu(mymenu, -1);
 pmydialog := GetNewDialog(mydialogid, nil, POINTER(-1));
 SetDialogItem(pmydialog, 4, Info^.PicNum);
 SetDialogItem(pmydialog, 5, Info^.PicNum - 1);
 fysize := Info^.nlines;
 fxsize := Info^.PixelsPerLine;
```

```
SetDNum(pmydialog, dxsize, fxsize);
SetDNum(pmydialog, dysize, fysize);
xsize := fxsize;
ysize := fysize;
xoffset := 0;
yoffset := 0;
border := 0;
SelIText(pmydialog, dborder, 0, 32767);
repeat
  ModalDialog(nil, itemhit);
 if itemhit = dxoffset then begin
   d := xoffset - GetDNum(pmydialog, dxoffset);
  if d <> 0 then begin
    border := 0;
     SetDNum(pmydialog, dborder, border);
    yoffset := GetDNum(pmydialog, dyoffset);
    ysize := fysize - yoffset;
    SetDNum(pmydialog, dysize, ysize);
    xoffset := GetDNum(pmydialog, dxoffset);
    xsize := fxsize - xoffset;
    SetDNum(pmydialog, dxsize, xsize);
  end;
 end;
 if itemhit = dxsize then begin
   xsize := GetDNum(pmydialog, dxsize);
   if (xsize + xoffset) > fxsize then begin
     xsize := fxsize - xoffset;
     SetDNum(pmydialog, dxsize, xsize);
  end;
 end;
 if itemhit = dysize then begin
   ysize := GetDNum(pmydialog, dysize);
   if (ysize + yoffset) > fysize then begin
     ysize := fysize - yoffset;
     SetDNum(pmydialog, dysize, ysize);
  end;
 end;
 if itemhit = dyoffset then begin
   d := yoffset - GetDNum(pmydialog, dyoffset);
  if d <> 0 then begin
    border := 0;
     SetDNum(pmydialog, dborder, border);
    yoffset := GetDNum(pmydialog, dyoffset);
    ysize := fysize - yoffset;
    SetDNum(pmydialog, dysize, ysize);
    xoffset := GetDNum(pmydialog, dxoffset);
    xsize := fxsize - xoffset;
    SetDNum(pmydialog, dxsize, xsize);
  end;
```

```
  end;
  if itemhit = dborder then begin
    d := border - GetDNum(pmydialog, dborder);
   if d <> 0 then begin
      border := GetDNum(pmydialog, dborder);
      xoffset := border;
      SetDNum(pmydialog, dxoffset, xoffset);
      yoffset := border;
      SetDNum(pmydialog, dyoffset, yoffset);
      xsize := fxsize + 2 * d;
      ysize := fysize + 2 * d;
      SetDNum(pmydialog, dxsize, xsize);
      SetDNum(pmydialog, dysize, ysize);
   end;
  end;
  if itemhit = 4 then begin
    m1 := GetCtl(pmydialog, 4);
   for i := 1 to nPics do begin
     if i = m1 then
       Inp1Info := pointer(WindowPeek(PicWindow[i])^.RefCon);
   end;
    fysize := Inp1Info^.nlines;
    fxsize := Inp1Info^.PixelsPerLine;
   if border > 0 then begin
      xsize := fxsize - border * 2;
      ysize := fysize - border * 2
   end
   else begin
      xsize := fxsize - xoffset;
      ysize := fysize - yoffset;
   end;
    SetDNum(pmydialog, dxsize, xsize);
    SetDNum(pmydialog, dysize, ysize);
  end;
  until (itemhit = ok) or (itemhit = cancel);
 if itemhit = cancel then begin
    DisposDialog(pmydialog);
    DeleteMenu(134);
    exit(ImageDiff);
 end;
  m1 := GetCtl(pmydialog, 4);
  m2 := GetCtl(pmydialog, 5);
  DisposDialog(pmydialog);
  DeleteMenu(134);
 for i := 1 to nPics do begin
   if i = m1 then begin
       Inp1Info := pointer(WindowPeek(PicWindow[i])^.RefCon);
      image1 := Inp1Info^.title;
   end;
```

```
  if i = m2 then begin
      Inp2Info := pointer(WindowPeek(PicWindow[i])^.RefCon);
     image2 := Inp2Info^.title;
  end;
 end;
 w1 := Inp1Info^.BytesPerRow;
 w2 := Inp2Info^.BytesPerRow;
 mx1 := Inp1Info^.PixelsPerLine;
 my1 := Inp1Info^.nlines;
 mx2 := Inp2Info^.PixelsPerLine;
 my2 := Inp2Info^.nlines;
 if (w1 > w2) or (my1 > my2) then begin
  beep;
  PutMessage('First image cannot be bigger than the second one');
    exit(ImageDiff);
 end;
 ShowWatch;
 {second image is the target we compare with}
 pc := 0;{percentage correct}
 pnc := 0;{percentage nearly correct}
 np := 0;{number of pixels compared}
 nzp := 0;{number of non zero pixels in first image}
 lt := 0;{number of pixels less than corresponding pixel in the target}
 gt := 0;{number of pixels greater than corresponding pixel in the target}
 me := 0;{mean error}
 mae := 0;{mean absolute error}
 mse := 0;{mean squared error}
 msne := 0;{mean squared normalised error}
 max := 0;{max pixel value in first image}
 min := 255;{min pixel value in first image}
 with Info^ do begin
    src1 := ptr(ord(Inp1Info^.PicBaseAddr) + w1 * yoffset);
    src2 := ptr(ord(Inp2Info^.PicBaseAddr) + w2 * yoffset);
   for i := 0 to ysize - 1 do begin
      BlockMove(src1, @Line1, mx1);
       src1 := ptr(ord4(src1) + w1);
      BlockMove(src2, @Line2, mx2);
       src2 := ptr(ord4(src2) + w2);
    c := xoffset;
    while c < (xsize + xoffset) do begin
      a := band(Line1[c], 255);
      b := band(Line2[c], 255);
     me := me + a - b;
    if a > b then begin
     gt := gt + 1;
     diff := a - b;
    if a > max then
    max := a;
     if b < min then
```

```
    min := b;
   end
    else if a < b then begin
     lt := lt + 1;
     diff := b - a;
    if b > max then
    max := b;
    if a < min then
    min := a;
   end
    else if a = b then begin {a=b}
     diff := 0;
    pc := pc + 1;
    if b > max then
    max := b;
    if b < min then
    min := b;
   end;
    if diff > 0 then begin
    if diff < 3 then
    pnc := pnc + 1;
   end;
    if a > 0 then
    nzp := nzp + 1;
    np := np + 1;
    mae := mae + diff;
      msne := msne + Exp(Ln(diff / 255) * 2);
      mse := mse + Exp(Ln(diff) * 2);
    c := c + 1;
   end;
 end;
 msne := msne / np;
  rms := sqrt(mse / np);
 pc := pc / np * 100;
 pnc := pnc / np * 100;
 mae := mae / np;
   NumToString(min, str1);
  str := concat(str, 'Min = ', str1, '     ');
  RealToString(mae, 1, 2, str1);
 str := concat(str, 'MAE = ', str1, '     ');
  NumToString(max, str1);
  str := concat(str, 'Max = ', str1, cr);
  NumToString(np, str1);
 str := concat(str, 'Pixels = ', str1, '       ');
  NumToString(nzp, str1);
 str := concat(str, 'Non zero pixels = ', str1, cr);
  NumToString(lt, str1);
 str := concat(str, 'Less than = ', str1, '   ');
  NumToString(gt, str1);
```

```
    str := concat(str, 'Greater than  = ', str1, cr);
    RealToString(msne, 1, 5, str1);
   str := concat(str, 'MSNE = ', str1, '      ');
    RealToString(rms, 1, 2, str1);
    str := concat(str, 'RMS = ', str1, cr);
    RealToString(pnc, 1, 2, str1);
    str := concat(str, 'Percentage nearly correct = ', str1, '%', cr);
    RealToString(pc, 1, 2, str1);
    str := concat(str, 'Percentage  correct = ', str1, '%', cr);

  end; {with}
 ShowMessage(str);
 ImageDiff := true;
end;


{  ------------  End of RHP user routines  ---------------------------}
procedure DoUserMenuEvent (MenuItem: integer);
 var
 ok: boolean;
begin
 case MenuItem of
  1:
  DoUserCommand1;
  2:
  DoUserCommand2;
  3:
  ok := GaussConvolve;
  4:
   ok := Sdn('', false);
  5:
  ok := ImageAdd;
  6:
   ok := ImageDiff;
  7:
   ok := Marhil;
  8:
   ok := Roberts('', false);
  9:
   ok := Threepeaks('', false);
  10:
   ok := Noise('', false);
 end;
end;

end.
```